CONTROL DE MOVIMIENTOS DE UN ROBOT UTILIZANDO MAPAS AUTOORGANIZADOS DE KOHONEN

Tesistas

Raúl Emilio Fernández
Oscar Andrés Sánchez

Director
Lic. Diego Bendersky

DEPARTAMENTO DE COMPUTACIÓN
FACULTAD DE CIENCIAS EXACTAS Y NATURALES
UNIVERSIDAD DE BUENOS AIRES

2007

En memoria de nuestros abuelos y tíos.

Índice

Agradecimientos 3
Resumen 4
Abstract 5
Capitulo 1 6
1.1 Introducción 6
1.2 Motivación 7
1.3 Soluciones al problema 8
1.4 Cinemática, dinámica 9
1.5 Alcance y Objetivos 10
Capítulo 2 12
2.1 Red Neuronal 12
2.2 Aprendizaje no Supervisado Competitivo 15
2.3 Feature Mapping 17
2.4 Red de Kohonen 18
2.5 Mapa Extendido de Kohonen o EKM (Extended Kohonen Map) 22
2.6 Redes Neuronales y Robótica 23
Capítulo 3 25
3.1 Estado del Arte 25
3.2 Trabajos Relacionados 26
3.2.1 Zalama E., Gaudiano P. and López Coronado J. 27
3.2.2 C. Versino, L.M. Gambardella 27
3.2.3 Kian Hsiang Low, Wee Khleng, Marcelo H. Ang, Jr 27
3.3 Método Propuesto 28
Capítulo 4 30
4.1 Arquitectura Utilizada 30
4.2 Método propuesto 32
4.3 Entrenamiento 33
4.4 Ejecución 36
4.4.1 Alcance del Objetivo 36
4.4.2 Evasión de obstáculos 41
4.4.3 Integración de Actividades 46
4.5 Situaciones Especiales 48
4.5.1 Objetivo 48
4.5.2 Obstáculos 49
Capítulo 5 51
5.1 El robot Khepera 51
5.1.2 Sensores de proximidad 52
5.1.3 Control de los motores 52
5.2 YAKS 53
5.2.1 Modificaciones realizadas al simulador 54
5.3 Software desarrollado 55
Capítulo 6 57
6.1 Análisis y Resultados 57
6.1.1 Entrenamiento 57
6.1.2 Alcance del Objetivo 60
6.1.3 Parámetros de configuración 62
6.1.4 Situaciones complejas 65
6.2 Conclusiones 70

6.3 Trabajo Futuro 71
Apéndice A 72
Archivos de configuración del simulador YAKS 72
Apéndice B 74
Configuración del Software desarrollado 74
Bibliografía 76

Agradecimientos

A Diego, nuestro director de tesis, por su experiencia, capacidad, humildad, paciencia y apoyo que nos ha brindado en este proyecto.

Oscar y Raúl

A mis padres: Carlota y Cacho (Américo); a mis hermanos: Julio y Sol (Soledad); a mis cuñados: Mariana e Ivan; a la alegría de la familia, mis sobrinos: Pilar, Sebastián y Laura; por todo el tiempo que no pude compartir en familia.

A mis tías y tíos: Muñe (Ester), Negro (Abdón), Kika (Francisca); a mi abuela: Isabel; a todos mis primos y parientes; quienes sumaron sus granitos de arena a lo largo de mis estudios.

A mis suegros: Elsa y Juan Carlos, por toda la ayuda que nos brindaron este año.

A Raúl, por tenerme infinita paciencia, por las garras que le puso a esta tesis, por haberme bendecido con su amistad, y por compartir conmigo días, noches y madrugadas de estudio y esfuerzo durante casi toda la carrera.

A mis ex-compañeros de estudios, sin los cuales jamás hubiera llegado a esta instancia: Aníbal T., Javier A., Oscar G., Hernan H., Nicolás B., Silvina D., Karina S., Jerónimo G., Carlos W., Estela B., Andrés T., Andrés F.

A mis amigos: Raúl, Mariela, Aníbal, Silvia, Pali (Pablo), Gabriel, Karina, Marcelo B., Marcelo M., Jerónimo, Diana.

A Karina, mi amada esposa, que recién casada tuvo que ponerse a lidiar con mis idas, vueltas y bajones sobre esta tesis, por brindarme su apoyo cuando más lo necesitaba, por arengar para que no aflojara, por guardar en secreto el último año de la tesis y por haberme ayudado a alcanzar esta meta.

Oscar

A mis papas, Esther y Enrique (quique), por todos los sacrificios que hicieron para que pudiera llegar hasta aquí. A mi hermano, Enrique por los fines de semana de ausencia.

A mis suegros Perla y Pablo, por las velas que han gastado en todos estos años. A mí cuñada Romina, por confiar en que llegaría.

A mis amigos Martín, Oscar, Jerónimo, Guille, Ulises, Karina K., Viviana, Carlos, Karina S, por su apoyo, compresión y por hacerme el aguante, quizás sin saberlo, en todo este proceso. Para aquellos que iniciamos nuestra amistad a través de esta carrera, por hacer tan placenteros estos largos años de estudio.

A Oscar porque sin él esta tesis jamás hubiese sido lo que es, por aguantar mis discusiones y los largos días de estudio.

Por último y más importante a mi esposa MARIELA, por su apoyo y aliento a pesar de todos los momentos en los cuales no pude estar, por el amor que me das sin el cual jamás podría haber llegado a este momento de mi vida.

Raúl

Y a todos los que nos han acompañado en este largo, pero muy largo camino, muchas gracias.

Oscar y Raúl

Resumen

Obtener los movimientos necesarios para que un robot pueda alcanzar un objetivo de forma autónoma, en un ambiente no estructurado y evitando obstáculos en el camino es esencial en robots móviles que operan en circunstancias adversas. Alcanzar una solución no es sencillo debido a que existen diversos aspectos que dificultan estas tareas, como lo son, lidiar con un ambiente dinámico, restricciones al movimiento del robot, presencia de ruido sistemático y no sistemático en el ambiente y en los sensores del robot.

En la presente tesis se ha desarrollado un controlador sensor – motor para permitir a un robot aprender sus movimientos autónomamente. Para lograrlo se utilizó una red neuronal conocida como Mapa Extendido de Kohonen. Nuestra propuesta logra que el agente alcance un objetivo aún en escenarios complejos, como por ejemplo: con obstáculos cóncavos, entornos reducidos, entornos con gran cantidad obstáculos, entre otros. Asimismo el agente consigue una trayectoria suave orientándose hacia el objetivo. La técnica desarrollada reduce la cantidad de parámetros necesarios, y no depende de las características del agente ni del entorno utilizado.

Abstract

Obtaining the necessary movements so that a robot can autonomously reach a target in a non-structured environment and avoiding obstacles in its way is essential in mobile robots that operate in adverse circumstances. To get a solution is not an easy task due the fact that there are various aspects that make those tasks difficult, such as dealing with a dynamic environment, restrictions to the movements of the robots, the existence of systematic and non-systematic noises in the environment and the sensors of the robot.

In this thesis, a control sensor - motor that allows a robot to learn its movements autonomously, has been developed. In order to get so, a neural network, known as Extended Kohonen Map, was used. Our proposal succeeds in making the agent reach a target even in complex scenarios, for instance, with concave obstacles, reduced environments, settings with a great number of obstacles, among others. Likewise, the agent manages to follow a smooth path orientated towards the target. The developed technique reduces the number of necessary parameters; and it depends neither on the characteristics of the agent nor on the environment used.

Capitulo 1

Esta sección identifica el dominio del problema y especifica los objetivos que han de alcanzarse.

1.1 Introducción

Uno de los objetivos de la robótica es el de construir sistemas inteligentes capaces de mostrar comportamientos racionales y complejos en la ejecución de tareas específicas, caracterizadas por la interacción física con un mundo real y dinámico a través de un cuerpo físico. Un robot autónomo es una máquina capaz de operar en un ambiente parcialmente desconocido e impredecible.

El campo de la robótica móvil tiene un amplio rango de posibilidades en la industria, incluyendo robots autónomos que limpien pisos en edificios o fábricas, sistemas de vigilancia, transporte de partes dentro de una industria sin necesidad de modificar las instalaciones, siembra y recolección de alimentos, entre otros.

Existe gran cantidad de investigaciones sobre robot móviles enfocadas en el problema de obtener los movimientos necesarios para que un robot pueda alcanzar un objetivo de forma autónoma, en un ambiente no estructurado y evitando obstáculos en el camino. Esto es esencial en robots móviles que operan en circunstancias adversas, como son la exploración del espacio, océano, áreas contaminadas. También lo es en áreas emergentes como robots de servicios, las cuales incluyen limpieza [HS/95], transporte, asistencia, etc.

Obtener una solución a esta tarea no es fácil, debido a que existen diversos aspectos que la dificultan, entre otros: lidiar con un ambiente no estructurado y dinámico, restricciones en los movimientos del robot, características no lineales en el sistema dinámico, presencia de ruido sistemático y no sistemático en el ambiente, en los sensores del robot y en los actuadores.

Sería entonces desafiante investigar y desarrollar una técnica robusta y flexible evitando estos apremios. Para hacerlo creemos que las respuestas pueden ser obtenidas de agentes biológicos que hoy en día realizan estas tareas: la capacidad de moverse con éxito en cualquier ambiente es imprescindible en la supervivencia de seres humanos y de animales.

Alguna vez cualquier persona ha intentado perseguir una mosca con sus manos y seguramente se ha preguntado como un insecto tan pequeño puede desarrollar tácticas de escape tan eficientes. Un investigador seguramente se ha hecho otro tipo de preguntas como ¿donde procesa la información visual que proviene de sus ojos? Los animales de todos los tamaños y tipos demuestran una capacidad notable para obtener comportamientos complejos con sistemas simples, pero ¿cuál es su secreto? Entender estos sistemas podría aportarnos una solución para utilizar en robots móviles.

El cerebro humano, por ejemplo, desarrolla una precisa coordinación entre lo percibido y los movimientos realizados de cara a muchos de los cambios que pueden presentarse en la inmediación de su cuerpo, y esta coordinación es obtenida en mayor parte sin un maestro externo.

Basados en estas consideraciones, un robot móvil puede entonces adquirir sus capacidades de movimiento mediante un aprendizaje similar. Actualmente los robots autónomos son controlados directamente por programas de control, por estimaciones de la cinemática inversa o por técnicas adaptativas. Estas últimas deben o bien ser adaptadas para un robot en particular o bien requieren de especificaciones técnicas del agente antes de ser utilizadas.

La tarea de navegación siguiendo un objetivo y evitando obstáculos, dada su complejidad, suele ser descompuesta, de manera jerárquica y modular, en tareas más sencillas o específicas. En un primer nivel, podemos determinar las tareas de autolocalización [BEF/96], construcción de mapas y planeamiento de camino o *path planning*. En un segundo nivel podemos encontrar las distintas técnicas y problemas relacionados con el control de motor. Este proyecto se enfoca en la implementación de un mecanismo de control de motor que permita al robot aprender sus movimientos autónomamente y lograr desplazarse hacia un objetivo [RF/95]. Esto permitirá desarrollar las bases fundamentales para el desarrollo de módulos de alto nivel.

La implementación de un controlador de motor eficiente debe seguir los siguientes criterios: el controlador de motor debe ser capaz de asociar la entrada proveniente de los distintos sensores y la salida producida, el robot debe ser capaz de producir un movimiento suave y preciso , debe poder alcanzar un objetivo en el mínimo tiempo posible y el algoritmo a desarrollar no debe estar ligado a un robot en particular.

Nuestro objetivo es obtener un controlador con estas características mediante aprendizaje no supervisado, esta noción será abordada en mayor detalle durante el desarrollo del presente trabajo.

1.2 Motivación

Muchos biólogos, psicólogos y entomólogos están interesados en el uso de robots móviles para validar estructuras de control observadas en el mundo biológico. A modo de ejemplo, podemos citar los trabajos de Franceschini, quien usó robots para validar la estructura de la retina observadas en las moscas [FPB/92], Beer, quien reprodujo el mecanismo que coordina el movimiento de las patas de los insectos al andar [EQCB/93], Deneubourg, quien obtuvo más información acerca del comportamiento colectivo de las colonias de hormigas [DGFSDC/90] . Estos son algunos de los estudios que se han realizan con robots, fuera de las aplicaciones industriales.

Tradicionalmente las investigaciones en robótica se han centrado en definir las interacciones en ambientes estáticos. Una gran parte de la robótica todavía se centra en los problemas concernientes a la manipulación de objetos y a los robots en estos ambientes. Sin embargo en estos últimos años se encuentran cada vez más investigaciones orientadas a los agentes móviles en ambientes dinámicos. El hecho de que el agente posea un cuerpo físico introduce nuevas variables: casi todos los robots están sujetos al deterioro, a la fricción, al mal funcionamiento y a otros factores físicos que nunca son tomados lo suficientemente en cuenta en una simulación. También existe la necesidad de que el agente perciba, actúe y muestre una serie de comportamientos racionales requeridos en ambientes dinámicos caracterizados por su variabilidad en el tiempo y el espacio.

Dentro de las investigaciones en robótica existen una gran cantidad de preguntas por responder que definen un amplio espectro de oportunidades y garantizan el futuro de las investigaciones en esta área.

1.3 Soluciones al problema

Como ya se ha dicho, en esta tesis se aborda el problema de alcanzar un objetivo, evitando obstáculos en el camino y si es posible habiendo elegido el camino más corto y sin movimientos bruscos.

Básicamente los modelos existentes pueden dividirse en tres grupos:

- Planeamiento: Estos métodos encaran el problema desde una visión global, generando un plan de movimientos a seguir desde un punto inicial hasta el objetivo deseado, para lo cual utiliza alguna forma de representación del ambiente.

- Evaluación del camino a seguir en forma local: en este tipo de métodos diferentes caminos a seguir son evaluados en cada paso del robot para determinar la ruta libre de colisiones y más efectiva para llegar al objetivo.

- Reactivo: En este caso, el robot elige la ruta a seguir basado en la información actual acerca de los obstáculos y la posición del objetivo. Este tipo de modelos se denominan también sistemas basados en sensores.

La mayoría de los controles de motor de un robot móvil son combinaciones de estos métodos utilizando planeamiento para determinar el camino a seguir y las otras técnicas para lidiar con obstáculos no conocidos y con la incertidumbre. Similarmente en los métodos de evaluación local se crea un plan con *checkpoints* u objetivos parciales: mientras el robot se mueve se generan estos puntos analizando un mínimo conjunto de posibles caminos en la vecindad del robot, libres de colisiones, basado en el sensado local.

Lo que estos métodos sacrifican es la optimalidad. El planeamiento local o los métodos reactivos, por ejemplo, pueden quedarse estancados en mínimos locales y requieren, para poder salir de esta situación, recalcular los pasos a seguir mediante planeamiento, sin embargo no necesitan un conocimiento total del entorno y no es necesario un gran poder de computo. El planeamiento global es demasiado costoso, el robot debe esperar hasta que el próximo paso a seguir sea determinado mediante algún algoritmo. Como ventaja de realizar un planeamiento, se puede decir, que al poseer un conocimiento del ambiente pueden analizarse una mayor cantidad de opciones para determinar el próximo paso.

Un buen algoritmo debe permitir desarrollar la habilidad de locomoción tanto de alto como de bajo nivel. En los niveles altos debe ser capaz de razonar acerca de la trayectoria que debe seguir y optimizar el largo del trayecto. En los niveles inferiores debe ser capaz de decidir sobre cada paso en la trayectoria del robot, esto es, debe proponer acciones que acerquen el robot hacia su objetivo mientras evita obstáculos en el camino.

Los niveles superiores utilizan mapas, imágenes de cámaras u otras representaciones complejas, de forma de generar una trayectoria libre de obstáculos, mientras que los niveles inferiores solo se basan en general, en la información recibida de sensores simples. Los niveles superiores pueden resolver hábilmente trayectorias mínimas o situaciones donde otros métodos pueden incurrir en mínimos locales, debido a que poseen una descripción global del ambiente. Sin embargo, cuando no se cuenta con información global, no pueden ser utilizados. También requieren un alto tiempo de procesamiento dado que evalúan diversas alternativas para seleccionar la mejor posible. Los sistemas basados en sensores no necesitan de esta información y esto ya es una ventaja por si sola, a su vez no son costosos computacionalmente ya que reaccionan a la información de los sensores, pero no consiguen caminos óptimos y pueden quedar atascados en mínimos locales. De todas formas es difícil programar un sistema de este tipo ya que se debe predecir todas las posibles situaciones y encontrar y analizar cada posible acción.

1.4 Cinemática, dinámica

El control reactivo es una técnica que asocia estrechamente la percepción y la acción generalmente en el contexto del comportamiento de las respuestas motoras, para producir respuestas puntuales en mundos dinámicos y desestructurados, sin el uso de representaciones abstractas o temporales [A/71].

El control reactivo se basa en el concepto de comportamiento, que proporciona la relación de los pares estímulo-respuesta para un determinado estado del entorno, haciendo que de la interacción de diferentes comportamientos con el entrono emerja la inteligencia necesaria para realización de determinadas tareas.

La actividad sensorial proporciona la información necesaria para satisfacer una respuesta a bajo nivel de un reflejo motor. Los modelos de comportamiento animal han servido como inspiración de diseño de comportamientos en sistemas de control reactivo.

El estudio de la coordinación entre los pares estímulo respuesta, lidian con una asociación bidireccional existente entre la percepción sensorial y el comando de motor ejecutado. Por un lado existe una transformación desde el espacio de las acciones de motor al espacio que resulta en distintas situaciones sensoriales: esta parte es conocida como *"forward association"* o cinemática directa [VG/95]. Por otro lado se necesita conocer cómo generar un control de motor para alcanzar un objetivo conociendo la información proveniente de los sensores: esta parte es conocida como *"inverse association"* o cinemática inversa. El problema en la cinemática inversa es que pueden existir múltiples soluciones para el mismo resultado final.

Con referencia a este problema, se ha estudiado intensamente la asociación existente entre sensores y motor en la coordinación de brazos robóticos [MRS/90] [WS/93] [KR/89]. Desde el punto de vista metodológico el estudio de esta coordinación puede realizarse de dos formas. El método clásico es asumir un modelo a priori de la interacción entre los sensores y los actuadotes, es decir un modelo basado en las leyes físicas, de esta forma la cinemática directa significaría predecir la posición del brazo robótico dada una cierta configuración, lo cual puede ser expresado mediante una ecuación matemática que represente la cinemática directa, mientras que el efecto final de posicionar el brazo puede ser expresado mediante una ecuación matemática que se

corresponda con la cinemática inversa. La opción alternativa consiste en la descripción de la interacción entre sensores y control de motor, para lo cual primero se obtienen datos del tipo (percepción, acción) mediante la observación del robot en ejecución. El primer modelo tiene como desventaja que depende estrictamente de las características físicas del robot y la dificultad de estimar la ecuación matemática correspondiente, el segundo modelo puede ser encarado mediante la utilización de redes neuronales [MRS/90] [WS/93].

La cinemática de un robot trata con el estudio analítico de la geometría del movimiento de un agente con respecto a un sistema de coordenadas de referencia fijo sin considerar las fuerzas o momentos que originan el movimiento. La dinámica, en cambio, trata con la formulación matemática de las ecuaciones del movimiento como un conjunto de ecuaciones matemáticas que describen la conducta dinámica del agente.

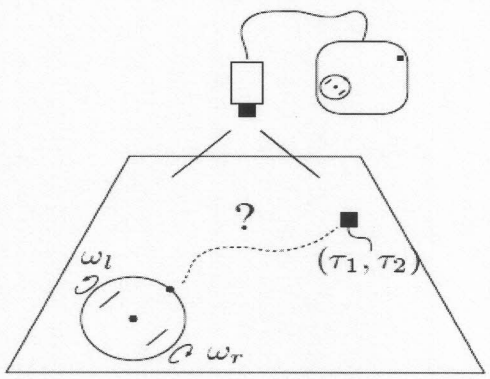


Figura 1.4.1: Representación grafica del problema.

1.5 Alcance y Objetivos

En la presente tesis se pretende lograr que un robot móvil adquiera sus capacidades de locomoción a través de un aprendizaje no supervisado, utilizando esencialmente una extensión de los mapas autoorganizados de Kohonen o EKM. Ambos conceptos serán explicados en detalle en los capítulos sucesivos. El método que se ha propuesto posee las siguientes ventajas:

- El controlador de motor es capaz de aprender la asociación entre la entrada de los sensores y la salida de control de motor sin un maestro externo.

- El robot aprende un repertorio de movimientos tan extenso como se desee, de esta forma se consigue realizar movimientos suaves reduciendo los errores de posicionamiento y permitiendo alcanzar un objetivo con un tiempo mínimo de demora.

- El método propuesto no depende estrechamente de las características físicas del robot, con lo cual diferencias físicas inclusive entre robots de la misma categoría pueden ser subsanadas, ya que cada robot aprende la forma en que puede realizar sus propios movimientos.

- El robot es capaz de evitar obstáculos con formas complejas en el camino hacia su objetivo

- El método propuesto aprovecha la información aprendida, evitando la necesidad de utilizar constantes ligadas al entrenamiento y ejecución de la locomoción.

Se ha desarrollado un software para el entrenamiento y observación de la red durante el aprendizaje y la ejecución de los comportamientos. A través de este sistema es posible configurar de manera *online* la forma en la cual el robot evitará los obstáculos y se acercará al objetivo, como también obtener información sobre la actividad neuronal y sensorial, entre otros.

Capítulo 2

Este capítulo presenta una introducción a las redes neuronales, prestando especial atención a los mapas autoorganizados de Kohonen.

2.1 Red Neuronal

Una red neuronal es un sistema formado por unidades de proceso que pueden verse desde el punto de vista matemático como funciones no lineales de E/S y desde el punto de vista computacional como procesadores.

También se define como una colección de procesadores paralelos conectados de manera tal que se ayuda a si misma a solucionar el problema que se está considerando. Son sistemas para el procesamiento de información, inspirados en el modo en el que las redes de neuronas biológicas del cerebro procesan información.

Una neurona es una célula que tiene cuerpo, un núcleo y una prolongación llamada axón que es la vía común para el contacto con otras neuronas. Luego existen otras conexiones más débiles llamadas dendritas. El mecanismo de intercambio se denomina *sinapsis*, y es un proceso electro químico: es disparado por una excitación eléctrica y luego desencadena un proceso químico y viceversa.

La forma en que las neuronas se conectan es lo que determina la arquitectura. Una red neuronal biológica no nace preprogramada con todos los conocimientos y habilidades que eventualmente tendrán. Un proceso de aprendizaje, que toma lugar sobre un periodo de tiempo, modifica la red para incorporar nueva información.

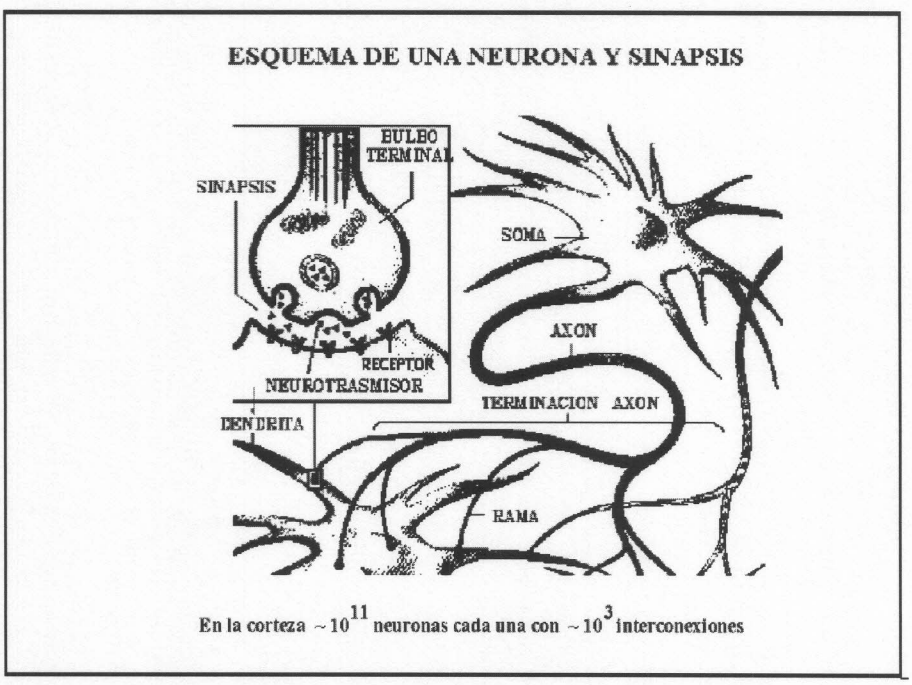


Figura 2.1.1: Esquema de una neurona biológica.

McCulloch y Pitts propusieron un modelo matemático de neurona, que puede apreciarse en la figura 2.1.2. A este modelo se le fueron introduciendo generalizaciones de forma de resolver distintos tipos de problemas.

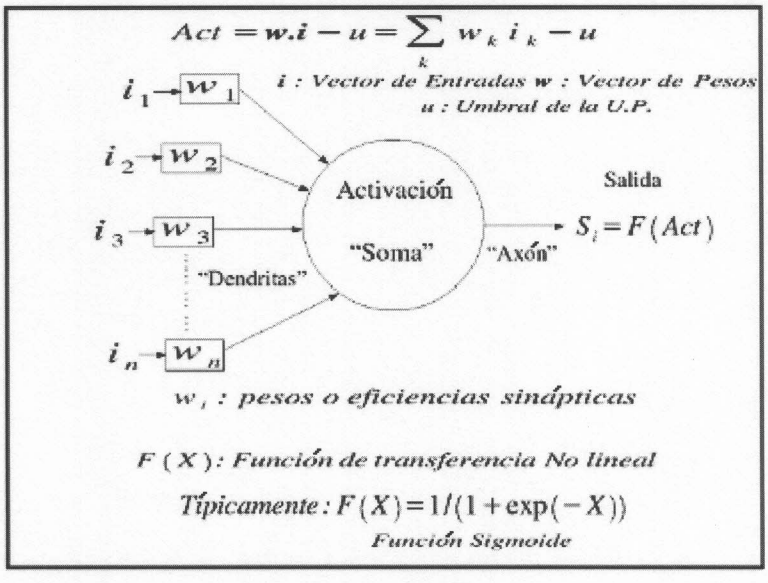


Figura 2.1.2: Ejemplo de una neurona artificial. Modelo de McCulloch y Pitts

Aprender en una red neuronal significa encontrar un conjunto apropiado de pesos. Esta tarea puede ser realizada de diferentes formas:

- Aprendizaje Supervisado: Se instruye la respuesta requerida para cada señal específica de entrada.

- Aprendizaje No Supervisado y Autoorganizado: La No Supervisión indica la ausencia de un maestro externo, el cual se compensa con la competencia y/o cooperación entre unidades vecinas por responder a un estímulo. Este proceso conduce a un "clustering" (agrupamiento) de la información según ciertas similitudes de sus rasgos. La red no contiene respuestas correctas o incorrectas, descubre por si sola categorías interesantes o ventajas en los datos de entrada.

El tipo de red neuronal utilizada en esta tesis utiliza un aprendizaje no supervisado. Las redes neuronales que emplean este modo de aprendizaje poseen tanto unidades de entrada como de salida pero a diferencia de las redes supervisadas, no reciben una realimentación del ambiente indicando como han de ser las salidas ni qué tan correctas son las mismas.

Las redes de esta clase deben "descubrir" por sí mismas patrones, características, regularidades, correlaciones o categorías en los datos de entrada y con ellas producir un código de salida.

Para lograr esta modalidad de cómputo es necesario entonces que las unidades y/o los pesos sinápticos sean capaces de autoorganizarse.

El aprendizaje no supervisado sólo es útil cuando existe suficiente redundancia en los datos; sin ella no sería posible encontrar patrones o características interesantes. En este sentido la redundancia es responsable del conocimiento.

Existen diversas posibilidades para la información que la salida de una red no supervisada puede proveer:

a) Familiaridad: Una salida única de tipo real podría indicar el grado de similitud que un patrón de los datos tiene con un patrón promedio típico concebido en el pasado por la red. Es decir la red puede gradualmente aprender qué es lo típico.

b) Análisis de Componentes Principales: Al extender el caso anterior a varias salidas involucra la especificación de una base de varias componentes (conjunto de ejes de referencia) a lo largo de los cuales se ejecuta la medida de similitud arriba mencionada. Un enfoque estadístico común a esto recibe el nombre de análisis de componentes principales.

c) Clustering (agrupamiento): Un conjunto de salidas binarias de las cuales sólo se activa una a la vez. Esta red podría indicarnos a cuál de diversas categorías pertenece un patrón de entrada dado. Las categorías involucradas han de ser determinadas por la red como consecuencia de las correlaciones en los datos de entrada. En este caso cada cluster o grupo de patrones similares se clasifica en una misma clase individual.

d) Determinación de Prototipos: Semejante al caso anterior pero la red suministra como salida un prototipo o ejemplar representativo de la clase. En este caso la red no supervisada funcionaría como una memoria asociativa pero las memorias son determinadas por la red a partir de los patrones de entrada y no son impuestas desde el exterior.

e) Codificación: La salida es una versión codificada de la entrada, generalmente de menor tamaño, pero conservando la mayor cantidad de información relevante posible. Esto es muy empleado como técnica de compresión de información; en este caso es necesario disponer de una red que ejecute el proceso inverso de decodificación.

e) Mapeo de Características: En el caso que las unidades de salida posean una disposición geométrica fija (ej.: una retícula cuadrada) y sean activadas una a la vez, entonces el sistema podría producir el mapeo (correspondencia) de los patrones de entrada con cada uno de estos puntos en la retícula. La idea es lograr un mapa topográfico de características en el cual patrones con características parecidos siempre activen unidades de salida vecinas en la grilla, de esta forma es de esperar la emergencia de una organización global de las unidades

Es claro que los casos anteriores no son necesariamente diferentes, y pueden ser combinados entre sí en diferentes formas: El resultado de codificación puede realizarse por ejemplo con análisis de componentes principales o con clustering, lo que recibe el nombre de Cuantización Vectorial. El análisis de componentes principales puede emplearse para lograr una reducción de la dimensión de los datos antes de proceder con

el clustering o el mapeo de características. De esta manera podrían simplificarse algunos problemas en la búsqueda de regularidades en datos.

Las arquitecturas involucradas en el modo de entrenamiento no supervisado son relativamente sencillas, las complejidades ocurren esencialmente en las reglas de aprendizaje. La mayoría de las arquitecturas consisten de una única capa de unidades, generalmente con un flujo de información unidireccional. La tendencia es que en esta clase de modelos la inspiración neurobiológica sea mucho más fuerte.

2.2 Aprendizaje no Supervisado Competitivo

En el caso del aprendizaje competitivo sólo una unidad (o una por grupo) se activa como respuesta a un estímulo. El entrenamiento ocurre a través de un proceso competitivo, en el cual cada unidad compite por el derecho de activarse bajo la acción de un estímulo externo.

Esta clase de unidades suelen ser referidas como "la ganadora toma todo" o "células abuela". En general el objetivo de estos dispositivos suele ser el agrupamiento o categorización de datos.

Un ejemplo de aplicación para este tipo de dispositivo no supervisado es la clasificación de patrones. En esta situación estímulos similares (patrones semejantes) han de ser clasificados en una misma categoría lo cual se establece bajo la activación de una misma unidad de salida. En el enfoque no supervisado, el dispositivo debe determinar las clases por sí sólo a partir de las correlaciones en los datos de entrada.

Este tipo de cómputo es básico en los procesos cognitivos de los seres biológicos y resulta de mucha utilidad y aplicación en la resolución de problemas de la vida real. Es útil en la codificación y compresión de datos, en la aproximación de funciones, procesamiento de imágenes, análisis estadístico y optimización combinatoria.

Otra clase de cómputo muy importante se obtiene cuando en el aprendizaje competitivo se incluye la relación geométrica entre las unidades, de esta forma es posible la organización de una relación geométrica particular, propia de los datos, en las unidades. Este proceso recibe el nombre de mapeo de características y consiste en el establecer correspondencia entre relaciones geométricas o topológicas de los patrones (en el espacio de patrones) con relaciones equivalentes en el espacio de las unidades es posible producir de esta manera mapas de características o de relaciones topológicas.

El dispositivo más simple de aprendizaje competitivo consiste de una sola capa de unidades cada una completamente conectada a un conjunto de estímulos de entrada $\{\xi_i\}$ a través de pesos sinápticos $w_{ij} \geq 0$.

En general se consideran tanto entradas como salidas binarias. En este dispositivo sólo una neurona de salida puede activarse bajo la presencia de un estímulo. Esta unidad, la ganadora, es normalmente aquella con el mayor potencial presináptico:

$$h_i = \sum_j w_{ij}\xi_j = w_i\xi$$

La unidad ganadora cumple con:

$$w_s \xi \geq w_i \xi \quad \forall i,$$

donde s define la unidad ganadora con salida unitaria. Lo que indica que la neurona ganadora es aquella unidad de vector de pesos que más se parece al patrón de entrada ξ.

No es muy relevante cómo se implementa en la computadora la característica de la "ganadora toma todo". Simplemente puede ejecutarse una búsqueda entre las unidades para determinar aquella con máxima activación. En una red real es posible implementar tal propiedad a través del uso de conexiones inhibitorias laterales entre las unidades de salida y una realimentación excitatoria de cada una. Seleccionando adecuadamente los pesos sinápticos laterales y de realimentación puede asegurarse lograr una única salida evitando oscilaciones.

Un dispositivo competitivo como el descrito implementa un clasificador de patrones según los criterios de la unidad ganadora. El problema que se presenta es el definir una regla de aprendizaje que permita la determinación de "grupos o clusters" sobre los datos de entrada, es decir la determinación de los pesos sinápticos w_s adecuados.

Como el procedimiento debe ser no supervisado, los pasos deben incluir:

a) Comenzar con pesos aleatorios pequeños. Es importante no "prejuiciar" a la red y romper cualquier simetría.

b) Presentar el conjunto de patrones de entrenamiento a la red, aleatoriamente o no. Alternativamente pueden ser seleccionados independientemente de una distribución *P* de patrones de entrada.

c) Para cada entrada se determina la unidad ganadora *s* entre todas las unidades

d) Se corrigen los pesos sinápticos w_{sj} de la unidad ganadora únicamente de tal manera de hacer que su vector de pesos se haga más parecido al patrón de entrada correspondiente. Este procedimiento aumenta la probabilidad de que en el futuro la unidad ganadora continúe ganando bajo la presentación de este patrón.

Una alternativa que se denomina Regla Estándar de aprendizaje competitivo es:

$$\Delta w_{sj} = \eta\left(\xi_j^{\mu} - w_{sj}\right)$$

Cuyo objetivo es acercar w_s directamente hacia ξ. Es de notar que esta regla de aprendizaje ha de aplicarse a los pesos de las unidades ganadoras.

Geométricamente puede visualizarse el efecto de la regla de aprendizaje competitiva como la búsqueda y localización de grupos (clusters) en los datos. Esto

ocurre ya que cada patrón de entrada compite por las unidades atrayendo hacia sí las unidades más cercanas. En el caso que los patrones de entrada se presenten con la misma frecuencia se alcanza un estado estacionario correspondiendo a un atractor estable.

Un problema que suele presentarse con el anterior esquema de aprendizaje es que unidades con pesos sinápticos bastante alejados de los patrones nunca puedan ganar en el proceso competitivo y por lo tanto nunca sean sometidas a entrenamiento. Unidades con estas características suelen ser referidas como unidades muertas. En la práctica puede ser deseable su ocurrencia para aquellos casos en los cuales el conjunto de patrones de entrenamiento es variable y se esperan patrones con rasgos diferentes en el futuro. De lo contrario se han ideado medidas para prevenir el problema:

a) La inicialización de los pesos puede realizarse con una muestra de los patrones mismos, asegurando así que estén en el dominio correcto.

b) Es posible corregir los pesos tanto de las unidades ganadoras como perdedoras. En este último caso con una tasa de aprendizaje menor. Así una unidad perdedora gradualmente se "alineará" en la dirección promedio lo que eventualmente puede conducir a que gane una competencia.

c) Si las unidades están dispuestas en un arreglo geométrico (rejilla bidimensional), entonces, pueden corregirse los pesos de las vecinas de las ganadoras también.

d) Se puede restar un término umbral u_i del potencial presináptico h_i ajustándolo de tal manera que unidades con frecuencia alta de perdedoras ganen. Unidades que son frecuentes ganadoras deben aumentar el u_i mientras que las perdedoras deben disminuirlo. Estabilizar este esquema puede ser algo complejo pero es realizable. Este mecanismo suele ser referido como de conciencia: Unidades que son ganadoras frecuentes se supone que sienten una especie de culpa y reducen su tasa ganadora. Con este método es posible lograr que m unidades de salida sean triunfadoras en promedio $1/m$ de las veces.

e) Se puede agregar ruido a los patrones de entrenamiento con la finalidad de distribuirlos mejor sobre el espacio de patrones.

2.3 Feature Mapping

En los párrafos anteriores no se ha hecho referencia a algún arreglo geométrico de las unidades. Sin embargo, si estas se distribuyen en una línea o en un plano es viable pensar en dispositivos en los cuales la posición de cada unidad pueda conllevar algún tipo de información y donde la cercanía de las unidades determine semejanza en tal tipo de información.

Por ejemplo si x_1 y x_2 son dos vectores de entrada y r_1 y r_2 las posiciones de las correspondientes unidades ganadoras, entonces r_1 y r_2 han de acercarse más a medida que x_1 y x_2 se hagan más similares (Mapa de Rasgos o Características).

Técnicamente el cómputo descrito constituye una correspondencia que preserva la topología (topology preserving map) desde el espacio de patrones de entrada a la línea o plano donde se localizan las unidades (Espacio de unidades). Una correspondencia de esta clase esencialmente lo que conserva es la relación de vecindad.

La idea parece trivial pero no lo es en razón de que pueden existir pares de espacios entre los cuales no es posible tal tipo de correspondencia. Es claro que el espacio de patrones ha de tener una métrica que defina la relación de vecindad.

Existen varias maneras de implementar un dispositivo de aprendizaje no supervisado que se autoorganice en un mapa de características:

a) Emplear aprendizaje competitivo agregando conexiones laterales en la capa de unidades. Estas han de ocurrir entre unidades vecinas y han de tener la forma de sombrero mexicano: excitatorias para las primeras vecinas e inhibitorias para las unidades más lejanas, con una intensidad que decae con la distancia. De esta forma las unidades más cercanas se especializan en patrones parecidos mientras que las más alejadas seleccionan su especialización en patrones diferentes.

b) Utilizar aprendizaje competitivo estándar, pero no sólo corregir los pesos de la unidad ganadora sino también la de sus unidades vecinas. Esta es la esencia del Algoritmo de Kohonen.

2.4 Red de Kohonen

Existen evidencias que demuestran que en el cerebro hay neuronas que se organizan en muchas zonas, de forma que las informaciones captadas del entorno a través de los órganos sensoriales se representan internamente en forma de mapas bidimensionales. Por ejemplo, en el sistema visual se han detectado mapas del espacio visual en zonas del córtex (capa externa del cerebro). También en el sistema auditivo se detecta una organización según la frecuencia a la que cada neurona alcanza mayor repuesta (organización tonotópica).

Aunque en gran medida esta organización neuronal está predeterminada genéticamente, es probable que parte de ella se origine mediante el aprendizaje: Esto sugiere que el cerebro podría poseer la capacidad inherente de formar mapas topológicos de las informaciones recibidas del exterior. De hecho, esta teoría podría explicar su poder de operar con elementos semánticos: algunas áreas del cerebro simplemente podrían crear y ordenar neuronas especializadas o grupos con características de alto nivel y sus combinaciones. En definitiva, se construirían mapas especiales para atributos y características.

A partir de estas ideas, Tuevo Kohonen presentó en 1982 un sistema con un comportamiento semejante. Se trataba de un modelo de red neuronal con capacidad para formar mapas de características de manera similar a como ocurre en el cerebro; el objetivo de Kohonen era demostrar que un estímulo externo (información de entrada) por sí solo, suponiendo una estructura propia y una descripción funcional del comportamiento de la red, era suficiente para forzar la formación de los mapas.

Este modelo tiene dos variantes denominadas LVQ (Learning Vector Quantization) y TPM (Topology Preserving Map) o SOM (Self Organizing Map). Ambas se basan en el principio de formación de mapas topológicos para establecer características comunes entre las informaciones (vectores) de entrada a la red, aunque difieren en las dimensiones de éstos, siendo de una sola dimensión en el caso de LVQ y bidimensional o tridimensional en la red SOM.

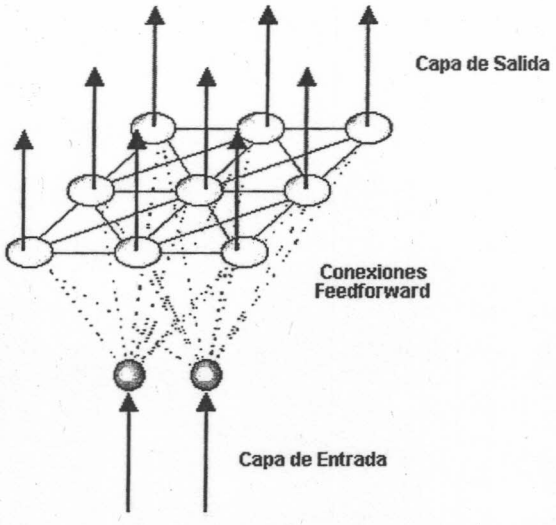


Figura 2.4.1: Conexiones en una red de Kohonen

El aprendizaje en el modelo de Kohonen es de tipo Off-line, por lo que se distingue una etapa de aprendizaje y otra de funcionamiento. En la etapa de aprendizaje se fijan los valores de las conexiones entre la capa de entrada y la salida. Esta red utiliza un aprendizaje no supervisado de tipo competitivo: Las neuronas de la capa de salida compiten por activarse y sólo una de ellas permanece activa ante una determinada información de entrada a la red. Los pesos de las conexiones se ajustan en función de la neurona que haya resultado vencedora.

Durante la etapa de entrenamiento, se presenta a la red un conjunto de datos de entrada (vectores de entrenamiento) para que ésta establezca en función de la semejanza entre los datos las diferentes categorías (una por neurona de salida), que servirían durante la fase de funcionamiento para realizar clasificaciones de nuevos datos que se presenten a la red. Los valores finales de los pesos de las conexiones entre cada neurona de la capa de salida con las de entrada se corresponderán con los valores de los componentes del vector de aprendizaje que consigue activar la neurona correspondiente. En el caso de existir más patrones de entrenamiento que neuronas de salida, más de uno deberá asociarse con la misma neurona, es decir pertenecerán a la misma clase.

En este modelo el aprendizaje no concluye después de presentarle una vez todos los patrones de entrada, sino que habrá que repetir el proceso varías veces para refinar el mapa topológico de salida, de tal forma que cuantas más veces se presenten los datos, tanto más se reducirán las zonas de neuronas que se deben activar ante entradas parecidas, consiguiendo que la red pueda realizar una clasificación mas selectiva.

El Algoritmo de Kohonen utiliza un "truco" computacional para lograr el efecto de la interconexión lateral tipo sombrero mexicano descripto en párrafos anteriores. No se implementan conexiones laterales como tales sino que se modifica la regla de

corrección de pesos de tal manera de involucrar relaciones de vecindad en el arreglo de unidades o neuronas.

La arquitectura de la red de Kohonen es un conjunto de unidades distribuidas sobre un arreglo uni-,bi- o tridimensional. Este dispositivo maneja patrones de entrada reales de dimensionalidad arbitraria. Todas las unidades se conectan completamente a las m componentes de los patrones de entrada a través de los pesos sinápticos correspondientes (o sea, cada unidad posee m pesos). Se aplica un algoritmo de entrenamiento competitivo en el cual para cada patrón de entrada se determina la unidad ganadora (unidad con vector de peso más cercano al patrón de entrada).

Siendo ξ el dato o patrón de entrada ingresado a la red y w_i el peso de la neurona i, la neurona ganadora s se determina según:

$$|w_s - \xi| \leq |w_i - \xi| \quad \forall i$$

donde la métrica utilizada es arbitraria aunque la más común es la Euclídea.

Una vez determinada la unidad ganadora se procede con la corrección de los pesos de la misma y de las unidades vecinas:

$$\Delta w_{ij} = \eta \Lambda(i,s)(\xi_j - w_{ij})$$

donde $\Lambda(i,s)$ es la función de vecindad que se centra en la posición $i = s$ y decae con la distancia $|r_i - r_s|$ entre las unidades i y s.

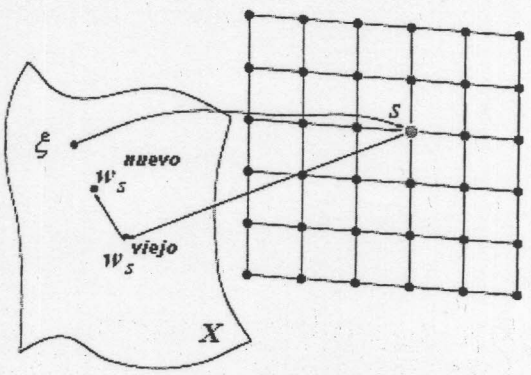


Figura 2.4.2: Proceso de aprendizaje de un Mapa de Kohonen

En el arreglo de las unidades una función típica empleada para este propósito suele ser una Gaussiana:

$$\Lambda(i,s) = \frac{1}{e^{\frac{|r_i - r_s|^2}{2\sigma^2}}}$$

σ es un parámetro que determina la amplitud de la campana de gauss.

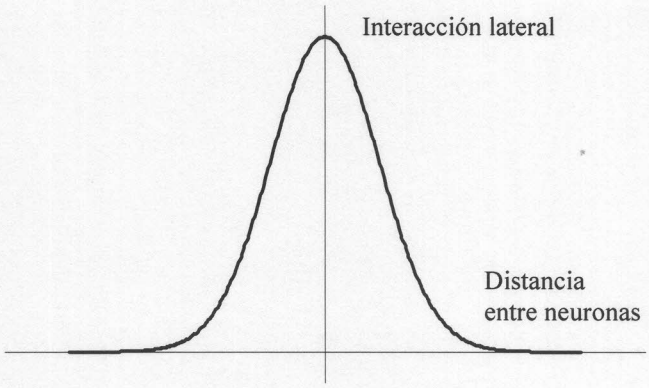


Figura 2.4.3: Función gaussiana empleada durante el aprendizaje.

La función gaussiana comienza por cubrir casi todo el espacio de neuronas y se reduce progresivamente hasta ser casi exclusivamente la neurona ganadora la que se actualiza. Para las cercanas a la ganadora, las actualizaciones se reducen exponencialmente según la distancia a aquella.

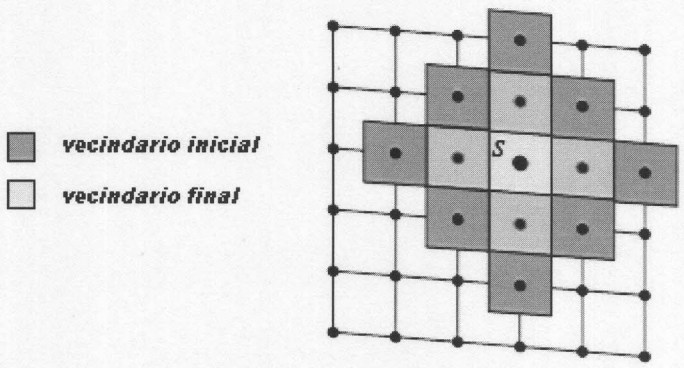


Figura 2.4.4: Vecindarios de la función gaussiana durante el proceso de aprendizaje

Se distinguen dos etapas en el período de aprendizaje

1. Ordenamiento: se realiza la organización topológica de los pesos.

2. Convergencia: las unidades se ajustan con precisión a los detalles de la distribución de las entradas.

En la primera, se entrenará a la red hasta que el vecindario se achique hasta el punto en que solamente este compuesto por la unidad ganadora.

En la segunda fase, la cual ocurre después que todos los patrones de entrenamiento han sido asignados a una unidad ganadora (aunque no necesariamente unidades diferentes), continuará el entrenamiento para un número arbitrariamente grande de iteraciones. Esto se hace para asegurar que la red se ha estabilizado, aunque no hay absoluta garantía de que esto ocurra.

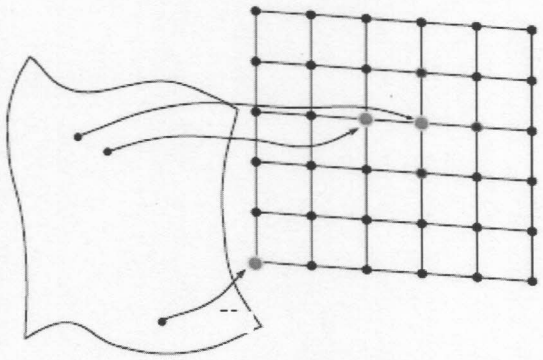


Figura 2.4.5: Datos similares del espacio de entrada son mapeados a lugares cercanos en la grilla

2.5 Mapa Extendido de Kohonen o EKM (Extended Kohonen Map)

Un EKM desde el punto de vista de la arquitectura es un mapa de Kohonen al cual se le ha adicionado a cada neurona *i* en la grilla o capa de salida un vector de pesos de salida *z* de forma de almacenar la salida de la neurona *i*.

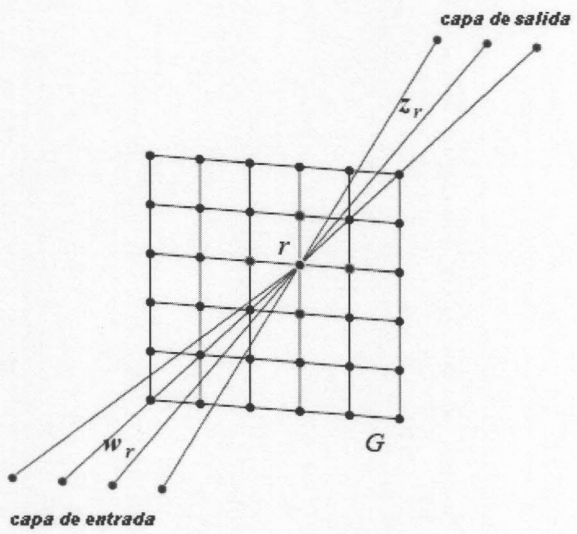


Figura 2.5.1: Arquitectura de un EKM

El aprendizaje en un mapa autoorganizado extendido se realiza siguiendo el procedimiento descripto a continuación.

- Se presenta a la capa de entrada un patrón *x*. Las neuronas en la grilla G compiten en respuesta al valor presentado a la red, al igual que en un mapa autoorganizado. La neurona *s* cuyo vector de pesos de entrada w_s se encuentre más cercano al patrón ingresado será la ganadora, y su vector de pesos de salida z_s es tomado como salida de la red ante el patrón *x*.

- Durante la fase de entrenamiento el patrón de entrada x y el patrón de salida y propuesto por un maestro, como valor de salida esperado, son aprendidos por la neurona ganadora y sus vecinas. Específicamente el vector de pesos de la neurona ganadora es ajustado de forma de acercarse al patrón de entrada x y su vector de pesos de salida es ajustado de forma de acercarse al patrón de salida y.

Este tipo de aprendizaje puede ser descripto como un entrenamiento competitivo-cooperativo. Competitivo porque las neuronas compiten a través de sus vectores de pesos de entrada para responder al patrón ingresado. Como consecuencia solamente la parte de la red involucrada con el patrón de entrada se encuentra afectada en la determinación de la neurona ganadora. La regla es también cooperativa ya que el vector de pesos de salida aprendido por la neurona ganadora es parcialmente asociado con los vectores de pesos de salida de sus vecinas.

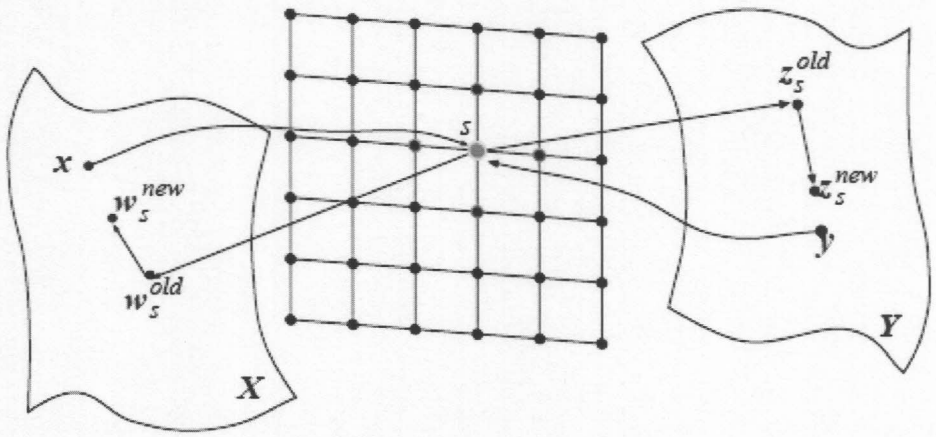


Figura 2.5.2: Proceso de aprendizaje de un EKM

2.6 Redes Neuronales y Robótica

Como se ha explicado en el primer capítulo, en la sección en la cual se define la cinemática y dinámica: dado un robot de una determinada forma y tamaño, el conocimiento para un movimiento suave puede ser obtenido y expresado a partir de un conjunto de pares <percepción, acción>. Cada par refiere a un paso unitario en la trayectoria del robot y describe una configuración dada de sensores y la acción seleccionada para ella. Más específicamente la percepción o configuración puede ser llevada a cabo con la lectura de una cantidad fija de sensores de proximidad, junto con la dirección y distancia del objetivo relativa al robot.

El par <percepción, acción> es la base para construir automáticamente sistemas basado en sensores, para este propósito puede ser utilizado un aprendizaje supervisado: el par <percepción, acción> es aprendido incrementalmente siguiendo los siguientes pasos:

- El sistema obtiene la "percepción" como entrada.

- Se genera una acción tentativa acorde al conocimiento actual
- Esta acción es comparada con la acción que debería haber sido realizada y la diferencia entre estas es usada para cambiar y mejorar el comportamiento del sistema

La secuencia de pasos es repetida para cada ejemplo disponible [RF/95] [GHRT03], utilizándose como sistema basado en sensores una Red Neuronal Artificial.

Como se puede observar el aprendizaje es difícil y no puede basarse solamente entrenando a la red neuronal con miles de ejemplos. Para facilitar el aprendizaje se puede tomar ventaja de una buena característica de los problemas de búsquedas de caminos, denominada *"similares percepciones requieren similares acciones" (similar perception require similar actions)*. Esto no es siempre verdadero pero si lo es en la mayoría de los casos, por ejemplo, dada una percepción de entrada, la red neuronal primero determina cual es la percepción más parecida según la experiencia de percepciones aprendidas hasta el momento, y segundo, se dispara la acción asociada a la percepción seleccionada en el primer paso. El primer paso es una operación de agrupamiento que apunta a agrupar similares experiencias dentro de una única percepción representativa. De esta forma percepciones similares contribuyen al aprendizaje de la misma (o similar) acción, mientras que percepciones diferentes (percepciones que requieren diferentes acciones) no interfieren unas con las otras dado que son mapeadas en diferentes percepciones representativas por el agrupamiento.

De lo mencionado puede deducirse que un Mapa Extendido de Kohonen puede ser entrenado de forma de aprender la asociación entre percepción y acción, manteniendo además que percepciones similares requieran acciones similares. Su utilización se explicará en mayor detalle a lo largo de esta tesis.

Capítulo 3

En este capítulo se detallará el estado del arte en la materia como así también se darán a conocer los métodos desarrollados que utilizan arquitecturas similares para resolver el problema del control reactivo en robots móviles.

3.1 Estado del Arte

Un controlador de movimiento reactivo de bajo nivel es usualmente utilizado como coordinador entre sensor y motor. Típicamente requiere dos comportamientos básicos: Alcanzar un objetivo y evitar obstáculos. Existen una gran variedad de métodos numéricos, basados principalmente en modelos físicos, para resolver estas dos tareas, como así también han sido objeto de estudio para distintos tipos de métodos de aprendizaje.

El controlador presentado en esta tesis posee una semejanza con un comportamiento de bajo nivel conocido como *homing* el cual predomina en la mayoría de los animales [WHW/93] [A/71] [RF/95] [C/96]. Este comportamiento puede ser definido como la habilidad de un robot autónomo de navegar desde un punto en particular hacia otro punto arbitrario en un ambiente específico. Esta habilidad es primordial para la adquisición de habilidades más complejas [TWBM/97]. Para adquirir este comportamiento de bajo nivel, un robot móvil debe ser capaz de aprender sus capacidades de movimientos o control de motor, es decir, debe aprender su dinámica en un entorno determinado a través de la coordinación entre sensores y motores.

Diversas investigaciones en este sentido han surgido de la investigación en brazos robóticos [TWBM/97] [JSG/95] [SGG/94] [WS/93] [MRS/90]. Los conceptos en este entorno pueden aplicarse al caso de un robot móvil, teniendo en cuenta las diferencias entre los dos tipos de robots.

Al igual que en otros trabajos, el método propuesto en esta tesis utiliza movimientos incrementales, es decir que el robot aprenderá los movimientos que puede realizar en un intervalo de tiempo, y para lograr llegar a su objetivo ejecutará los movimientos aprendidos evaluando en cada paso el próximo movimiento a ejecutar.

Para determinar la correlación entre los sensores y motores puede utilizarse modelos basados en las leyes físicas, la cinemática directa (control de motor a sensores) es utilizada en las técnicas de localización y autoposicionamiento conocidas como *dead reckoning*[1]. Basados en la información obtenida del robot, como el desplazamiento producido, la posición momentánea relativa a una posición inicial conocida, puede ser calculada utilizando ecuaciones matemáticas simples [BEF/96]. Estos métodos no pueden ser utilizados para grandes distancias debido a que el modelo de cinemática posee ciertas inexactitudes, los actuadores tienen una precisión limitada y fuentes externas afectan en la locomoción del robot, como el resbalamiento de las ruedas. Esto provoca que el error de posicionamiento se acumule con el tiempo. La cinemática inversa (sensores a control de motor) también puede representarse por ecuaciones matemáticas [ZGL/95] pero sufre de los mismos problemas. Estos modelos denotan un

[1] Refiere al proceso de la actualización y estimación de una determinada posición basado en el conocimiento del tiempo, velocidad y dirección del movimiento.

comportamiento ideal, los cuales sólo se encuentran en ambientes de simulación: en el mundo real el movimiento del robot diverge de este comportamiento con el tiempo.

El problema entonces se transfiere a encontrar un método que produzca resultados semejantes pero más cercanos al mundo físico. Las investigaciones en este sentido se orientan a la obtención de datos a través de observar el comportamiento del robot, es decir, la asociación entre sensores y motor durante la navegación del agente en un entorno. De esta forma no es necesario un conocimiento sobre la cinemática y dinámica del robot, y puede entonces utilizarse las correlaciones observadas para estimar los parámetros necesarios. Este problema de aproximación es de hecho un proceso de aprendizaje.

El proceso descripto puede ser realizado en forma supervisada o no supervisada. Las desventajas de un aprendizaje supervisado recaen en que es necesario el conocimiento de un maestro, el cual es utilizado durante la etapa de entrenamiento. De existir el conocimiento de un maestro, el aprendizaje puede ser deducido en forma directa, es decir el maestro debe indicar para cada situación como debe resolverse, este modelo es relativamente fácil de implementar. Una debilidad de estos métodos es que el comportamiento aprendido depende sensiblemente de la distribución y la secuencia de sucesos del conjunto de entrenamiento. Otro de los problemas es que no permite la adaptación a distintos ambientes, ya que al cambiar el entorno todo el proceso debe ser repetido para el nuevo entorno. Por el contrario un aprendizaje no supervisado puede mitigar estos problemas logrando que un robot puede adaptarse fácilmente al entorno donde navega [LKA/04] [LKA/03].

Diversos algoritmos de aprendizaje no supervisado han sido utilizados en la coordinación de robot manipuladores y robots móviles, entre los que se incluye los algoritmos genéticos [POS/07] [HMB/98], algoritmos basados en reglas [YB/96] [S/97], fuzzy logic [S/97], redes neuronales artificiales [ZGL/95] [RF/95] [VG/95] [VG/00] [WS/93] [SGG/94] [JSG/95] [SR/97] [MRS/90] [KE/94] [CWJGL/95] [LKA/04] [LKA/03] y aprendizaje por refuerzo [K/02] [HK/00] [HEK/99].

3.2 Trabajos Relacionados

Para resolver el problema de la coordinación entre sensor y motor diversos métodos han sido propuestos [ZGL/95] [RF/95] [VG/95] [VG/00] [LKA/04] [LKA/03]. Todos buscan el mismo objetivo, el cual es aprender autónomamente la cinemática y/o dinámica inversa de un robot móvil. La presente tesis sigue los lineamentos originales de K.J. Schulten [WS/93] [MRS/90] [WS/93] utilizando una extensión de los mapas autoorganizados de Kohonen (Extended Kohonen Map o EKM).

Se puede observar inmediatamente las ventajas de utilizar un SOM para esta tarea: En primer lugar se realiza una reducción del espacio de entrada, y en segundo lugar el proceso de computo puede ser realizado sobre un conjunto representativo del espacio de entrada. Luego, al introducir nuevos datos a la red, éstos pueden ser mapeados de forma similar sin la necesidad de realizar nuevamente todo el proceso. Este tipo de red ha sido utilizada en diversos campos, como lo son visión y análisis de imágenes, robótica, procesamiento de datos, lingüística y sistemas de inteligencia artificial, entre otros [KKK/98].

3.2.1 Zalama E., Gaudiano P. and López Coronado J.

Zalama, Gaudiano y Coronado desarrollaron un controlador para un robot móvil denominado NETMORC [ZGL/95] el cual aprende autónomamente el control de motor de un robot móvil. La red esencialmente se compone de dos vectores de neuronas que codifican la distancia y el ángulo del desplazamiento de un robot, y una grilla de dos dimensiones que codifica las velocidades de las ruedas. Cada neurona de los dos vectores están totalmente conectadas a las neuronas de la grilla. Durante la fase de aprendizaje se generan velocidades aleatorias en las ruedas y se observa el desplazamiento obtenido por el robot, y luego cada par velocidad-desplazamiento es asociado con la grilla y los vectores de distancia y ángulo. Los pesos de la neurona ganadora en los vectores son entrenados de acuerdo a una regla de aprendizaje denominada regla de Grossberg's. En la fase de testeo la red entrenada es utilizada para obtener las velocidades en las ruedas necesarias acorde a la distancia y ángulo hacia el objetivo; si el objetivo se encuentra fuera del espacio aprendido por la red neuronal, entonces se realizan los movimientos incrementales necesarios para poder alcanzarlo. Al final de cada movimiento el sistema de sensores actualiza la posición del robot con respecto al objetivo de forma de que el siguiente movimiento pueda ser establecido. Este controlador ha sido testeado en un simulador e implementado en un robot real.

3.2.2 C. Versino, L.M. Gambardella

Versino y Gambardela [VG/95] [VG/00] demostraron la utilización de un SOM de dos dimensiones para aprender el control de movimiento de un robot. Cada neurona en el SOM codifica un control de motor y un desplazamiento. Durante la etapa de aprendizaje un conjunto de ejemplos de entrenamiento son recolectados generando velocidades aleatorias en las ruedas, moviendo al robot y observando el desplazamiento producido. El control de motor en cada ejemplo es utilizado para encontrar la neurona ganadora acorde a la neurona con control de motor más "cercano" al introducido. Luego, los pesos de este nodo y las neuronas vecinas son actualizados acorde a la regla de aprendizaje. Se puede observar que el SOM es entrenado mediante la dinámica directa, es decir desde los controles de motor hacia el espacio de desplazamientos. En la etapa de testeo el SOM es utilizado mediante la dinámica, es decir, para mover el robot al objetivo designado, la distancia y el ángulo del robot hacia el objetivo sirven como entrada a la red neuronal y la red devuelve el control de motor necesario para dirigir el robot hacia el objetivo. Si el objetivo se encuentra fuera del espacio de acción, se adopta la misma metodología que el método explicado anteriormente. Este método ha sido testeado en un simulador y en un robot Khepera.

3.2.3 Kian Hsiang Low, Wee Khleng, Marcelo H. Ang, Jr

El método propuesto por Low, Khleng y Ang [LKA/04] [LKA/03] utiliza la misma representación que el SOM propuesto por Versino y Gambardela, pero el entrenamiento es realizado mediante dinámica inversa, es decir, se utilizan como vectores de entrada los desplazamientos obtenidos mediante la observación del robot luego de ejecutar un control de motor. Los métodos descriptos anteriormente realizan una discretización del espacio de entrada en un grupo de regiones disjuntas, cada una gobernada por una neurona la cual esta fijada a priori con una topología regular. Esto conlleva a que para lograr un menor error de autoposicionamiento o realizar movimientos más finos, se debe contemplar la suficiente cantidad de neuronas. En cambio Low, Khleng y Ang

proponen una topología irregular, la cual toma forma automáticamente durante los movimientos realizados por el robot en su entorno. Esto tiene como efecto obtener zonas mas densas en los desplazamientos que han sido ejecutados mayoritariamente y zonas menos densas en movimientos que rara vez han sido ejecutados.

Al iniciar el entrenamiento, los vectores de control de motor generados son incorrectos, pero a medida que el robot se mueve, la red se autoorganiza usando los vectores de control de motor y los correspondientes desplazamientos obtenidos mediante el control de motor ejecutado.

La característica más relevante del método propuesto es que propone la utilización de un conjunto o ensamble de SOM denominado EKM cooperativo, de forma de que no solo el robot aprenda sus movimientos durante el proceso de alcanzar un objetivo, sino que también pueda evitar obstáculos en el camino. Los primeros en utilizar una arquitectura de similares características fueron Walter and Ritter. La red propuesta se denominó PSOM jerárquico (Parameterized Self-Organizing Maps) y su utilización estuvo destinada a aprender el movimiento de un brazo robótico bajo diferentes contextos. PSOM es una variante que puede aprender con un conjunto reducido de datos de entrenamiento y sin embargo mantiene buena exactitud. Para hacer esto, PSOM realiza una interpolación sobre un mapeo continuo múltiple utilizando un conjunto reducido de funciones predefinidas e independientes básicas y vectores de pesos construidos directamente de los datos de ejemplos. Para asegurar buena interpolación, los datos son ordenados topológicamente para obtener los vectores de pesos. Para un PSOM jerárquico cada PSOM es entrenado separadamente, resultando en diferentes valores de pesos para diferentes PSOM [WR/95] .

El EKM cooperativo posee tres módulos:

1. Módulo de Alcance del Objetivo: constituido por un EKM para la localización del objetivo que es activado por la presencia del objetivo dentro del rango de alcance sensado.

2. Módulo de Evasión de Obstáculos: constituido por *n* EKMs para la localización de obstáculos, los cuales son activados por la presencia de obstáculos dentro del rango de alcance que sensa cada uno.

3. Módulo de Integración Neuronal: constituido por un EKMs de control de motor, que tiene el rol de interfase, integrando las señales de actividad desde los EKMs por competición y cooperación, produciendo señales de motor apropiadas.

La propuesta fue testeada mediante simulación, implementar el algoritmo propuesto en robots reales se establece como trabajo futuro.

3.3 Método Propuesto

El controlador de motor que se ha desarrollado utiliza las ideas enunciadas en los párrafos anteriores: se buscó obtener una arquitectura sencilla pero a su vez conseguir resolver algunos de los problemas que se presentan en las demás soluciones. La propuesta de Low, Khleng y Ang [LKA/04] [LKA/03], es la primera en utilizar los mapas

autoorganizados no sólo para lograr aprender la dinámica del robot sino para llevar a cabo este tipo de tareas. El inconveniente con la arquitectura presentada en estos trabajos es que utiliza un conjunto de EKMs con la consiguiente necesidad de mayor nivel de procesamiento y espacio para almacenar las redes neuronales. Más importante aún, el algoritmo de ubicación de objetivo y detección de obstáculos está basado en funciones gaussianas las cuales son evaluadas utilizando parámetros que están ligados directamente con el robot y el entorno en el cual se ejecuta el procedimiento. Como consecuencia estos parámetros determinan sensiblemente el comportamiento del agente.

El método propuesto tiene como primer objetivo eliminar la necesidad de estos parámetros sin afectar el mapeo entre los movimientos reales que puede conseguir el robot y los desplazamientos aprendidos durante el entrenamiento, basado a su vez en el aprendizaje de la mayor cantidad de movimientos posibles, logrando conseguir un camino hacia el objetivo de forma precisa. Para lograr esta meta se ha desarrollado una técnica por medio de la cual cada neurona determinará un valor que representa una actividad neuronal con respecto a una situación dada, para lo cual se tomó ventaja de la organización topológica de las neuronas en el mapa autoorganizado, que distintas propuestas hasta el momento no han considerado, evitando de esta forma la utilización de funciones susceptibles a los valores de parámetros y ligadas a un robot y entorno en particular.

El EKM utilizado en la arquitectura presenta en su capa de entrada los controles de motor y en su capa de salida los desplazamientos. Se ha diferenciado el entrenamiento de la ejecución: durante el entrenamiento se utiliza y aprende la dinámica directa mientras que en la ejecución de lo aprendido se utiliza la dinámica inversa. Los vectores de pesos asociados a las neuronas poseen información adicional: dentro de esta información se encuentra el tiempo durante el cual se ejecuta un control de motor y la orientación que posee el robot luego de haber efectuado un desplazamiento respecto a su posición anterior. Se han propuesto funciones de distancia que tomen en cuenta esta información de tal forma que la neurona ganadora seleccionada en cada paso del robot sea aquella que en base a las funciones de distancia implementadas se corresponda con un desplazamiento que logre llevar al agente hacia su objetivo, evitando obstáculos. La premisa principal es orientar el robot hacia su objetivo pero sin dejar de establecer movimientos suaves, con lo cual aún existiendo movimientos que posicionen al agente a menor distancia del objetivo, un movimiento que logre un desplazamiento más suave dejándolo a una distancia similar será el ejecutado.

Esta información adicional nos permite elegir movimientos que no solo garanticen movimientos suaves sino que también prioricen el hecho de que sus desplazamientos sean de tal forma de que el robot se mueva hacia delante. Como los sensores de detección de obstáculos del robot utilizado se encuentran en mayor cantidad en el frente del robot, orientar al robot hacia el objetivo hace factible la localización de obstáculos que de otra forma pueden no ser encontrados.

El agente ha sido dotado de una funcionalidad que permite que habiéndose estancado en obstáculos complejos pueda salir airoso de los mismos. La propuesta presentada ha sido evaluada por medio de pruebas en simulación, trabajos sobre robot reales deben ser testeados en investigaciones futuras.

Capítulo 4

Este capítulo presenta la arquitectura y el detalle del algoritmo utilizado por el controlador de motor propuesto.

4.1 Arquitectura Utilizada

La propuesta presentada en esta tesis utiliza un único EKM para aprender la dinámica de un robot, es decir el mapeo entre los controles de motor y los desplazamientos que puede realizar el agente mediante la aplicación de este control de motor. La capa de entrada a este EKM estará definida por los controles de motor y la capa de salida por los desplazamientos, de forma de obtener un mapeo desde el espacio de controles de motor hacia el espacio de desplazamientos.

Se asume que existe alguna forma de obtener información sobre los desplazamientos producidos por el agente, ya sea mediante una cámara, sistema de posicionamiento, cálculo de la odometría, u otro. En esta tesis los datos requeridos son brindados por un simulador.

Luego, pueden observarse la distancia d y ángulo α en radianes, que definen el desplazamiento como la posición final relativa a la posición y orientación inicial del agente al ejecutar un paso. Este desplazamiento se ha de notar como $u = (\alpha, d)$ donde $d \in \Re$ (la magnitud de los valores es determinada por el simulador) y $\alpha \in [-\pi, \pi]$. Además se cuenta con información sobre la orientación del robot, denotándose a ésta como θ , donde $\theta \in [-\pi, \pi]$ y representa la orientación final del robot respecto de la orientación inicial, luego de haber ejecutado un desplazamiento u. En la figura 4.1.1 (a) puede observarse como se determina estos parámetros a través de una cámara.

En esta tesis se decidió ampliar el concepto de desplazamiento incorporando la orientación del robot, con lo cual $u = (\alpha, d, \theta)$. Esta información será de suma importancia en nuestro procedimiento, el cual será explicado en mayor detalle luego, en este capítulo.

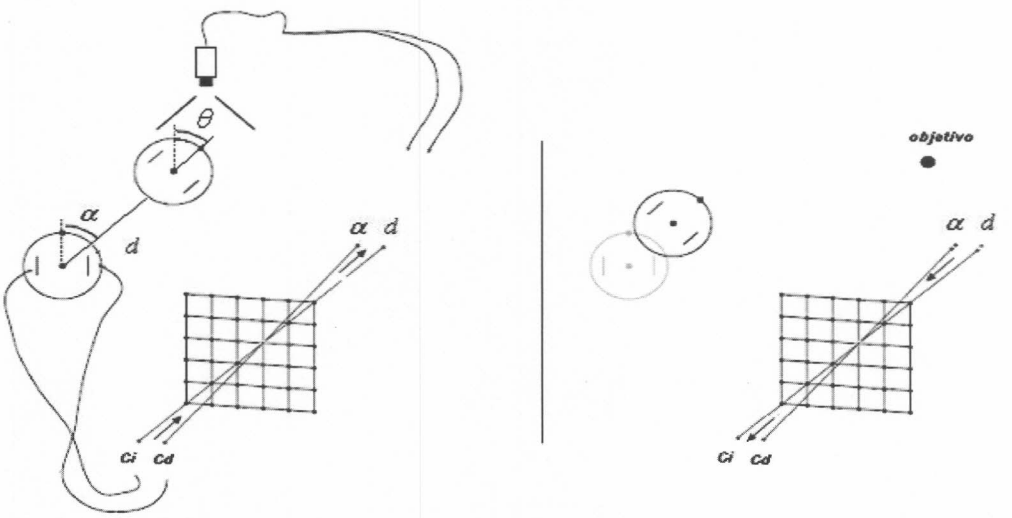
Asumiendo que el robot posee dos ruedas, se define el control de motor $C = [Ci, Cd, t]$, donde Ci denota la velocidad de la rueda izquierda, Cd la de la rueda derecha y t representa la duración en milisegundos de la ejecución de dichas velocidades, $Ci, Cd \in [C_{\min}, C_{\max}]$ y $t \in N$. El vector de pesos de entrada de la neurona r esta definido por $w_r = [Ci_r, Cd_r, t_r]$.

Las neuronas que conforman el EKM poseen como vector de pesos de entrada los controles de motor que puede producir el agente, por lo cual el peso de la r-esima neurona resulta en $w_r = [Ci_r, Cd_r, t_r]$, que codifica las velocidades en ambas ruedas y el tiempo de duración de estas velocidades. La capa de salida considera los desplazamientos y orientaciones del robot que ha de producirse con los correspondientes controles de motor, es decir el peso de salida en la neurona r corresponde al vector $z_r = [d_r, \alpha_r, \theta_r]$ donde $d_r \in [0, d_{\max}]$, $\alpha_r \in [-\pi, \pi]$, $\theta_r \in [-\pi, \pi]$.

En este trabajo hemos diferenciado el entrenamiento de la ejecución: Durante el entrenamiento se utiliza y aprende la dinámica directa (control de motor – desplazamiento), mientras que en la ejecución de lo aprendido se utiliza la dinámica inversa (desplazamiento – control de motor).

En la figura 4.1.1 puede observarse ambas etapas, en la figura (a) se ejemplifica el entrenamiento del EKM, donde el control de motor ejecutado es ingresado a la red y asociado con el desplazamiento producido por el mismo. En la figura (b) se ejemplifica la ejecución del agente, tras la determinación de la ubicación del objetivo se utiliza la red, como se verá mas adelante, de forma de obtener el control de motor que lleve al agente a la posición deseada, acercándolo hacia el objetivo.

La localización del objetivo puede realizarse por medio de algún mecanismo similar al utilizado para la observación del desplazamiento del agente. Entre estos pueden incluirse una cámara de video, un sistema de emisión de señales (sonoras, de radiofrecuencia, etc) ubicado en el objetivo o algún sistema de posicionamiento. En este trabajo esta información es brindada por el simulador utilizado.



(a) (b)

Figura 4.1.1: (a) – Entrenamiento del EKM, (b) – Ejecución: utilización de dinámica aprendida de forma de dirigirse hacia el objetivo.

Se decidió tomar en la capa de entrada a los controles de motor en lugar de los desplazamientos volviendo a las versiones originales del mismo [VG/95] [VG/00], ya que hacerlo de forma inversa [LKA/04] [LKA/03] puede producir que no se logre una coordinación desplazamiento / control de motor. Esto sucede debido a que existen desplazamientos que pueden ser producidos por controles de motor totalmente diferentes que no se encuentran cercanos en la capa de salida, con lo cual no se cumple que similares percepciones produzcan similares acciones, objetivo buscado mediante la utilización del EKM.

En la figura 4.1.2 puede observarse cómo producir un giro completo a la derecha, un giro completo a la izquierda o quedarse inmóvil producen el mismo

desplazamiento del robot, y sin embargo las velocidades en las ruedas que fueron aplicadas son completamente diferentes.

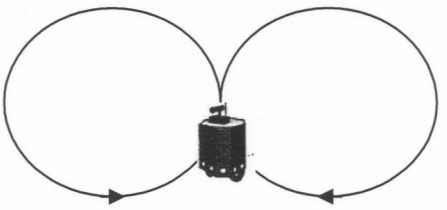


Figura 4.1.2: Tres controles de motor diferentes producen el mismo desplazamiento, *(Ci=8, Cd=4), (Ci=4, Cd=8), (Ci=0, Cd=0).*

El hecho de agregar información adicional como la orientación y el tiempo a los pesos de las neuronas reviste en la importancia de no solo aprender la dinámica del robot sino también de obtener información acerca de que tiempo es el que le lleva al robot conseguir ese desplazamiento en base al control de motor $C = [C_i, C_d]$ ejecutado y como lo deja orientado el mismo. Este punto es de suma importancia en el método propuesto ya que por medio de esta información se determinará la ejecución de un movimiento suave y preciso, a la vez que se amplía el conjunto de movimientos posibles. Esta observación se explicará en detalle más adelante.

4.2 Método propuesto

El método que proponemos se basa en obtener un mecanismo de forma tal que el robot se vea atraído hacia un objetivo, y que la presencia de obstáculos inhiba los movimientos no convenientes. Esta atracción o inhibición provocada por el objetivo y los obstáculos puede ser medida mediante señales, las cuales denominaremos *actividades*. Estas señales deben combinarse de tal forma de obtener el control de motor más conveniente para el agente.

De esta apreciación se deriva que el EKM es utilizado no sólo para aprender la dinámica del robot, sino que la información aprendida será utilizada para obtener los valores de actividad neuronal.

Se entiende, entonces, el término *actividad* como un valor de excitación o inhibición que presentará cada neurona ante la presencia de una entrada dada. Para el caso del objetivo, se ingresará como entrada a la red la posición $u = (\alpha, d)$ del mismo, relativa a la posición y orientación del agente. Se evaluará, luego, al EKM de forma tal que cada neurona devuelva como salida un valor excitatorio de actividad.

El robot posee sensores de proximidad que pueden determinar la ubicación de un obstáculo. Luego, por cada sensor que ha detectado un obstáculo, se evaluará a la red ingresando como entrada la posición del obstáculo $u_j = (\alpha_j, d_j)$ detectada por el sensor *j*, obteniendo nuevamente como salida la actividad neuronal que produce la presencia de obstáculos, pero a diferencia del objetivo, la actividad para este caso es una señal inhibitoria.

Una vez obtenidas las actividades en base a la ubicación del objetivo y la detección de obstáculos, se produce una sumatoria de dichas actividades, donde la actividad generada por la posición de los obstáculos detectados por cada uno de los sensores, resta a la actividad generada por el objetivo, de forma de que la neurona que posea mayor actividad luego de esta operación, será la indicada para ejecutar su control de motor. Esta neurona será aquella que haga dirigirse al agente hacia el objetivo logrando que el mismo no produzca colisiones con los obstáculos detectados. La siguiente figura hace referencia al procedimiento descripto, mostrando gráficamente los pasos involucrados en el mismo.

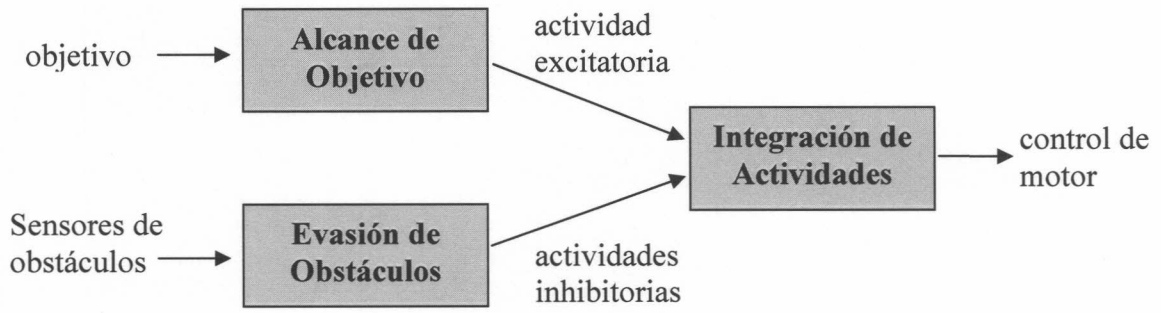


Figura 4.2.1: Esquema general de la etapa de ejecución del método propuesto.

Como ha de apreciarse una de las dificultades del método propuesto radica en el cálculo de las actividades y la forma en la cual la red es entrenada. Las secciones siguientes explican en detalle este procedimiento.

4.3 Entrenamiento

Como se ha podido observar en el capítulo dedicado a las redes neuronales y en especial a los mapas autoorganizados, un mapa de Kohonen, en este caso un EKM, replica en la posición de las neuronas la distribución de los datos de entrada, con lo cual si la red es entrenada con una distribución no uniforme de controles de motor, existirá un conjunto de controles de motor que no podrá ser aprendido adecuadamente. La idea entonces es lograr entrenar a la red con una distribución uniforme y con un conjunto de controles de motor que represente la mayor cantidad de movimientos distintos que puede ejecutar el agente.

Realizar el aprendizaje on-line, es decir, mientras se lleva a cabo la tarea de avanzar hacia el objetivo [LKA/04] [LKA/03], tiene como inconveniente que los ejemplos con los cuales se entrenará a la red se irán restringiendo a los desplazamientos ejecutados durante el proceso de avanzar hacia el mismo. El agente durante su etapa de ejecución ejecuta en la mayoría de los casos sólo controles de motor que lo hacen avanzar hacia delante, por lo cual el conjunto de ejemplos con los cuales será entrenada la red es reducido. Para mejorar este desempeño pueden aplicarse técnicas para explorar aleatoriamente otros movimientos posibles cada cierta cantidad de pasos. Igualmente, adicionando este tipo de métodos, la capacidad de movimientos continúa limitada. Debe notarse que sólo aprendiendo cierto tipo de movimientos, no es posible lograr un buen control del movimiento del agente, ya que en caso de que el robot deba

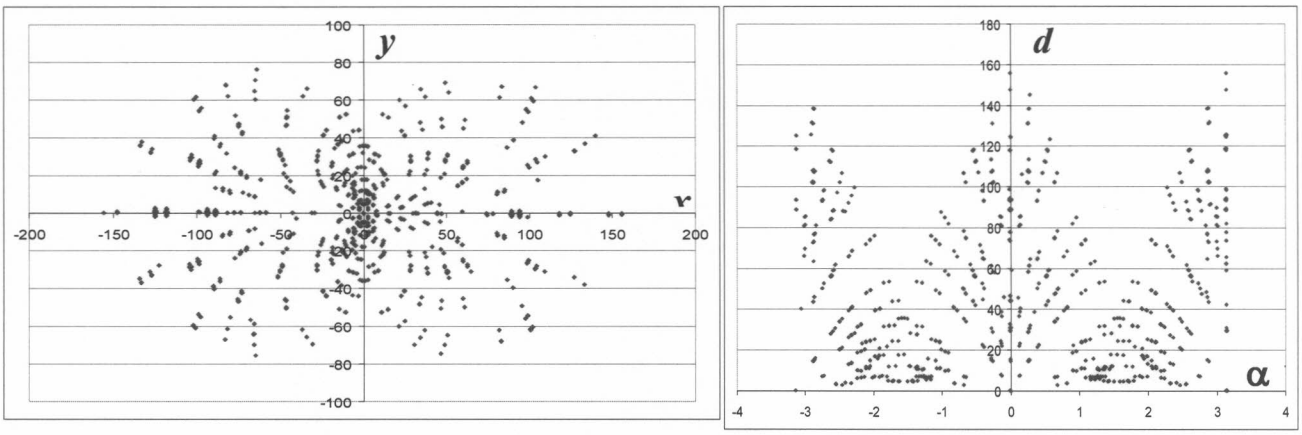
ejecutar, durante la interacción con el entorno, un movimiento que no se corresponde con los movimientos aprendidos, y la acción obtenida puede no ser la esperada.

Otro de los problemas que ocasiona un entrenamiento on-line es la extensa cantidad de tiempo necesario para que la red se estabilice.

Si la cantidad de movimientos con los cuales se entrenará a la red, resultan en un conjunto reducido, la red ajustará los pesos de las neuronas de forma tal que existirá una zona de mayor cantidad de neuronas cada una de las cuales con un desplazamiento similar a la de su vecina. Podría pensarse que esto es bueno porque se consigue una mayor cantidad de movimientos distintos que hacen que el robot se mueva hacia adelante. Sin embargo esto no es real: Muchas de las neuronas dentro de esta zona provocarán exactamente el mismo movimiento, ya que el robot produce velocidades en las ruedas con valores discretos y por lo tanto no puede asociar correctamente el desplazamiento correspondiente a estas neuronas con los controles de motor que puede ejecutar. Además se logra una supuesta motricidad fina en una determinada zona acotada pero no se podrán obtener movimientos suaves en amplias zonas del espacio.

Como se desea que la red neuronal aprenda la mayor variedad de desplazamientos, se ha utilizado para esta tesis un entrenamiento off-line, es decir, se han ejecutado en forma aleatoria controles de motor observándose los desplazamientos producidos. La red es entrenada con este conjunto y luego del entrenamiento toma lugar la ejecución de la tarea encomendada al robot móvil. Realizar el entrenamiento fuera de línea no limita el método propuesto en la presente tesis en cuanto a su capacidad de adaptación al entorno. El robot puede modificar lo aprendido, ya que durante la locomoción del agente en un entorno determinado, el entrenamiento puede seguir produciéndose, utilizando para la regla de aprendizaje, un vecindario disminuido de la neurona ganadora y una velocidad de aprendizaje de menor magnitud. Con lo cual en el caso de que el robot cambie su dinámica, ya sea por un desperfecto, alteración de tiempo en los actuadores de las ruedas o porque el entorno ha cambiado, se ajustarán los pesos de las neuronas a esta situación aprendiendo la nueva dinámica resultante.

En la figura 4.3.1 se puede observar 10.000 desplazamientos reales, que han sido producidos por un robot Khepera, el cual varía las velocidades en sus ruedas en el rango [-10, 10] utilizando el simulador YAKS. Las velocidades en las ruedas fueron aplicadas con un $\Delta t=2$ segundos. En la figura (a) pueden observarse las distintas posiciones en el espacio x-y, que el robot logró alcanzar ejecutando movimientos en forma aleatoria. La figura (b) muestra la misma información en coordenadas polares.



(a) (b)

Figura 4.3.1: Desplazamientos reales producidos en forma aleatoria por el robot

El algoritmo para entrenar la red se detalla a continuación:

1. Se obtiene en forma aleatoria un control de motor $C_p = [Ci_p, Cd_p, t_p]$ a ejecutar.

2. El control de motor es ejecutado observándose el desplazamiento $V_p = [d_p, \alpha_p, \theta_p]$ producido.

3. El control de motor es introducido como entrada al EKM

4. Se obtiene la neurona ganadora s acorde al control de motor ingresado, utilizando para cada neurona i la siguiente función de distancia:

$$d_i = D(w_i, C_p) = (Ci_p\, t_p - Ci_i\, t_i)^2 + (Cd_p\, t_p - Cd_i\, t_i)^2,$$

donde la neurona ganadora es la de menor distancia en la red neuronal y w_i es el peso de salida de la neurona i.

5. Se ajustan los pesos de entrada w de las neuronas i en el vecindario Nw_g de la neurona ganadora s

$$\Delta w_i = \eta_z\, G(i, s)\, (C_p - w_i)$$

donde $G(i, s)$ es una función gaussiana sobre la distancia en nodos entre la posición de la neurona i y s en la grilla del mapa autoorganizado, y η_w es la velocidad de aprendizaje utilizada en la campa de entrada de controles de motor.

6. Se ajustan los pesos de salida z de las neuronas i en el vecindario Nz_s de la neurona ganadora s

$$\Delta z_i = \eta_w\, G(i, s)\, (V_p - z_i)$$

donde $G(i, s)$ es la función gaussiana enunciada en el punto anterior y η_z es la velocidad de aprendizaje utilizada sobre la capa de salida de desplazamientos.

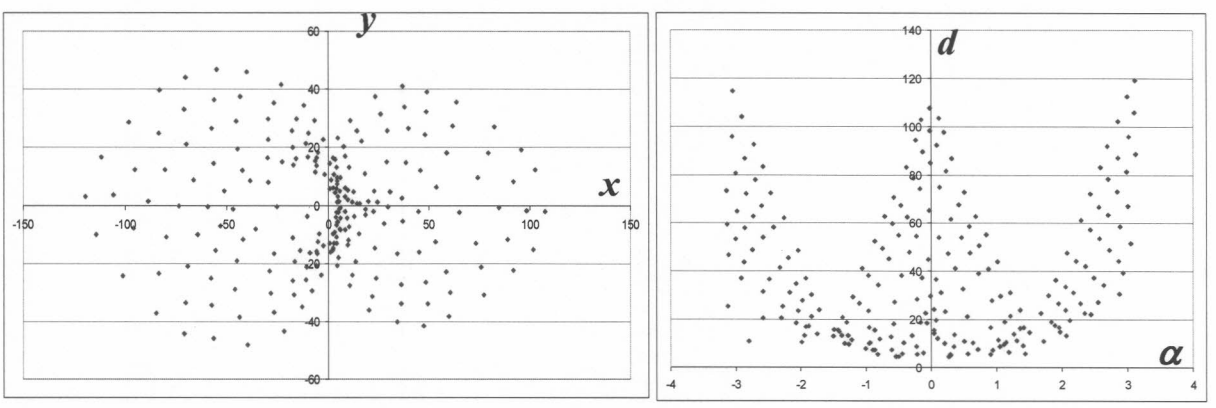
La resta $(V_p - z_i)$ no refiere la resta de vectores, sino que esta definida de la siguiente forma:

$$(V_p - z_i) = (d_p - d_i, Fa(\alpha_p - \alpha_i), Fa(\theta_p - \theta_i))$$

donde Fa es una función que corrige el valor de un ángulo al intervalo $[-\pi, \pi]$. Luego $z_i = z_i + \Delta z_i$, a esta suma también se le aplica Fa sobre el ángulo y la orientación.

En cada paso del algoritmo de aprendizaje los pesos de la neurona ganadora s y sus vecinos son actualizados, y la magnitud de las modificaciones es proporcional a $G(i, s)$. Luego de que la autoorganización se ha producido, para cualquier neurona ganadora se determina un control de motor C que produce el desplazamiento deseado $V = V_s$.

En el siguiente gráfico puede apreciarse la red neuronal resultante, mediante la distribución de las neuronas según los desplazamientos que ha logrado aprender cada una de ellas, luego de entrenar el EKM según el procedimiento descripto, ejecutando los 10.000 ejemplos graficados en la figura 4.3.1. La distribución (a) se corresponde con los desplazamientos de cada neurona en el espacio x-y, (b) muestra la misma información expresada en coordenadas polares. De esta forma puede observarse en que posición quedaría ubicado el robot respecto a su posición inicial, si ejecuta el control de motor asociado por medio de la neurona.



(a) (b)

Figura 4.3.2: Mapa autoorganizado extendido, correspondiente al mapa de desplazamientos una vez entrenada la red mediante el algoritmo descripto.

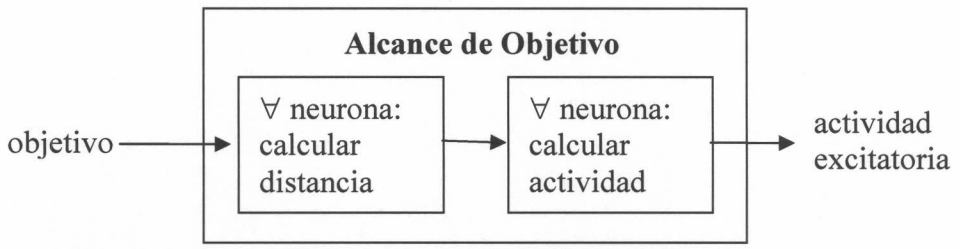
4.4 Ejecución

4.4.1 Alcance del Objetivo

La localización del objetivo se realiza durante la ejecución de la tarea requerida al robot, utilizando la red neuronal ya entrenada, para lo cual en cada paso el agente sensa el entorno y determina la localización del objetivo, ingresando esta información a la red. La entrada entonces, es el desplazamiento determinado por la ubicación del

objetivo respecto de la posición y orientación del agente, ya que lo que se busca es obtener una solución para la dinámica inversa. Como resultado final se obtendrá un valor de actividad por cada neurona.

Para obtener los valores de actividad, se ha definido una función de distancia que será evaluada entre la posición del objetivo ingresada a la red y el vector de pesos de desplazamientos que contiene cada una de las neuronas. Estas distancias serán los parámetros de entrada para el cálculo de la actividad.



Cálculo de distancias

El cálculo de distancia se basa en una suma ponderada sobre las componentes ángulo, distancia y orientación, de tal forma de priorizar algunas de las componentes sobre las demás, como se explicará más adelante.

Parece natural dar mayor prioridad al ángulo, de modo de que la primera intención del robot sea girar de forma de orientarse hacia el objetivo antes que avanzar. La distancia se hace realmente relevante cuando el robot se acerca al objetivo y debe reducir la velocidad de sus ruedas gradualmente. Esta proposición es mencionada también en otras propuestas [LKA/04] [LKA/03], en las cuales se realiza el cálculo de la función de distancia como un promedio ponderado, dándole mayor importancia al ángulo que a la distancia. En el método propuesto en esta tesis se considera, además, la orientación del robot, ya que la idea es procurar que el robot produzca un control de motor que lo lleve no solamente a un punto más cercano al objetivo sino orientado hacia el mismo. Notar que la mayor cantidad de sensores se encuentran en el frente del robot, por lo cual avanzar hacia adelante hará que el agente perciba mejor su entorno.

No necesariamente el punto más cercano será el desplazamiento ideal que debe ejecutar el robot: Desplazarse hacia un punto menos cercano pero mejor orientado hacia el mismo provocará que en el siguiente paso se pueda conseguir un movimiento mas exacto y preciso, además de requerir menos cambios de dirección para conseguir su meta final.

En la figura 4.4.1 se puede apreciar cómo desde una posición inicial existen dos posiciones a las cuales desplazarse (a) y (b). Los métodos actuales utilizarían la posición (b) como desplazamiento a producir, ya que posee el mejor ángulo. Sin embargo la posición (a) a similar cercanía al objetivo, logrará que el robot mejore su comportamiento en el siguiente paso logrando un movimiento suave. En cambio se puede apreciar que la trayectoria conseguida al utilizar el desplazamiento (b) resulta un poco brusca.

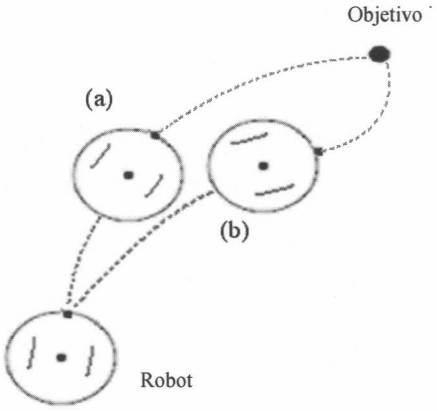


Figura 4.4.1: Alternativas de movimiento de un robot.

La función de distancia debe considerar que las componentes utilizadas en el cálculo de la suma ponderada tienen diferentes escalas. Considerar, por ejemplo, que mientras α está limitada a un rango de valores $[-\pi,\pi]$, la distancia al objetivo d puede ser arbitrariamente grande.

Se define el algoritmo de cálculo de distancias para con el objetivo de la siguiente forma:

1. Ingresar al sistema la ubicación del objetivo: $u_t = (\alpha_t, d_t)$

2. Calcular para toda neurona i la distancia entre el valor ingresado y el vector de pesos de salida (desplazamientos) de la neurona, utilizando la función de distancia definida a continuación:

$$D(z_i,u_t) = d_i = \beta_\alpha (Fa(\alpha_t - \alpha_i))^2 + \beta_d \left\| (\alpha_t,d_t) - (\alpha_i,d_i) \right\|^2 + \beta_\theta (Fa(\alpha_t - \theta_i))^2$$

[EQ 4.4.1]

donde Fa es una función que corrige el valor de un ángulo al intervalo $[-\pi,\pi]$, y β_α, β_d, β_θ son los parámetros que determinarán la importancia que se otorgará al ángulo, distancia y orientación.

Se debe notar que no es la función de distancia la que determina la neurona ganadora que posee el control de motor que debe ser ejecutado: La actividad neuronal generada a través de estas distancias será integrada con la actividad neuronal que generará cada uno de los sensores de obstáculos, y el valor de más alta actividad será luego el que indicará la neurona que posee el control de motor a ejecutar.

Con las distancias calculadas para todas las neuronas, se calcula la actividad neuronal que cada neurona establecerá para con la ubicación del objetivo como se define a continuación.

Cálculo de actividad del objetivo

El método propuesto por Low, Khleng y Ang [LKA/04] [LKA/03] utiliza para el cálculo de actividades neuronales funciones gaussianas centradas en la neurona

ganadora del mapa autoorganizado. De esta forma, la actividad de mayor valor se centra en la neurona de menor distancia al objetivo, y disminuye los valores de actividad en forma de campana de gauss para las neuronas cercanas topológicamente a la misma. El problema de esto reside en la dificultad de estimar los valores para los parámetros de las funciones gaussianas y lo fuertemente ligados que se encuentran estos valores a las características propias del robot, escalas utilizadas y características del entorno en el cual el robot se desplaza.

Nuestro método propone utilizar la información que se encuentra disponible en el mapa autoorganizado de forma de construir una función de actividad que se aproxime de la mejor forma posible a una representación de valores de excitatoriedad acordes a la ubicación del objetivo. La ecuación [EQ 4.4.1] es utilizada como medida de distancia entre el desplazamiento que cada neurona representa y la ubicación del objetivo. Puede entonces evaluarse la red para todas las neuronas y mapear los valores de distancia obtenidos con los valores de actividad asociados.

Se asume que los valores de actividad se encuentran en el rango *[0, 1]*, y luego se realiza el siguiente procedimiento:

1. Por cada neurona i calcular el valor de distancia d_i según la ecuación [EQ 4.4.1], obteniéndose el vector de distancias D:

$$D = (d_1, d_2, \ldots, d_{i-1}, d_n) \quad d_i \geq 0 \; \forall i$$

2. Obtener, el máximo y mínimo valor:

$$d_{\min} = \min(D), d_{\max} = \max(D)$$

3. Calcular la actividad A para cada neurona i:

$$A_i = \begin{cases} 1 \quad si \quad d_{\max} = 0 \\ \dfrac{(-d_i + d_{\max} + d_{\min})}{d_{\max}} \quad si \quad d_{\max} > 0 \end{cases}$$

Se consigue entonces que la función de actividad posea la misma distribución de valores que las distancias asociadas con cada neurona, haciendo que la actividad sea máxima para valores pequeños de distancia y mínimo para valores de distancia grandes, evitando la utilización de más parámetros y logrando que la función no se encuentre restringida a características propias del robot y su entorno.

Las siguientes pantallas fueron obtenidas del software desarrollado y el simulador utilizado. En las mismas se observa la actividad resultante de haber evaluado la red en la circunstancia que puede apreciarse en el escenario en la figura 4.4.2, donde el objetivo se encuentra ligeramente hacia la derecha del robot y la línea dentro del cuerpo del robot indica la orientación que posee el mismo. Se observan además los gráficos correspondientes a la actividad resultante conseguida mediante el método que hemos propuesto. La figura 4.4.3 (a) corresponde a la actividad de las neuronas en el mapa autoorganizado; los colores más intensos representan valores de actividad más elevados. La figura 4.4.3 (b) corresponde a los desplazamientos que posee asociados cada neurona, vistos en coordenadas cartesianas (pantalla inferior) y en coordenadas

polares (pantalla superior). La neurona ganadora es la indicada para ejecutar el control de motor asociado, el cual hará que el agente se ubique en la posición que puede observarse en el mapa de desplazamientos.

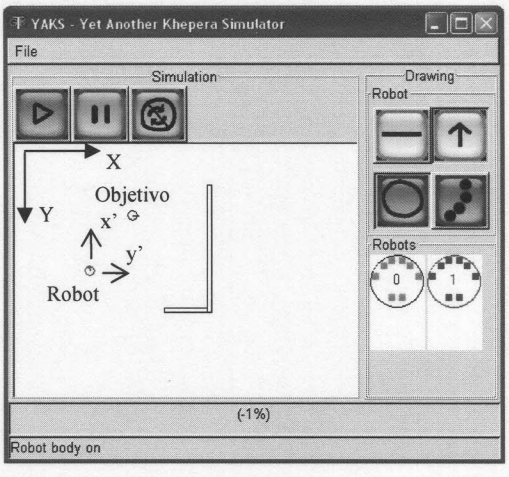
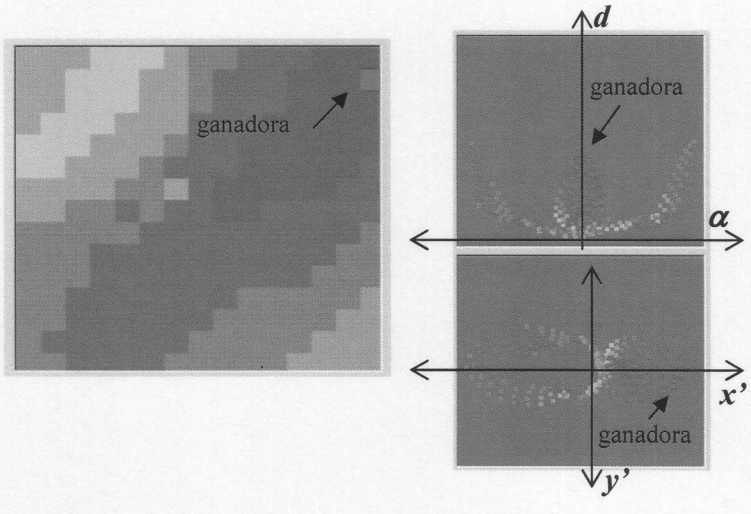


Figura 4.4.2: Escenario del simulador YAKS. Los ejes x' e y' corresponden al sistema de coordenadas relativos a la posición y orientación del robot.



(a) (b)

Figura 4.4.3: (a) actividad neuronal en el mapa autoorganizado. (b) desplazamientos asociados con las neuronas del EKM, en coordenadas polares y cartesianas.

Para poder observar mejor la actividad resultante a continuación se observa el gráfico de la misma.

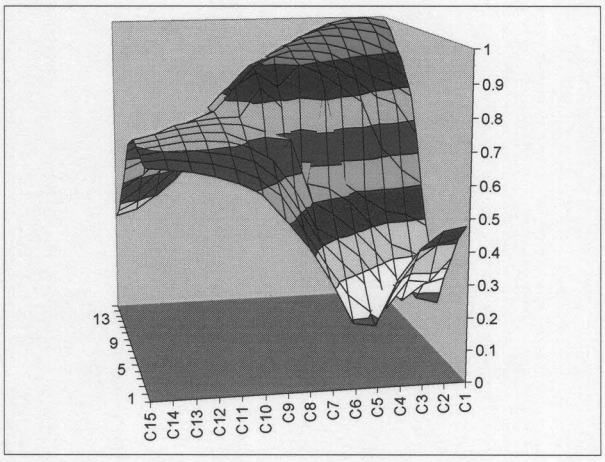
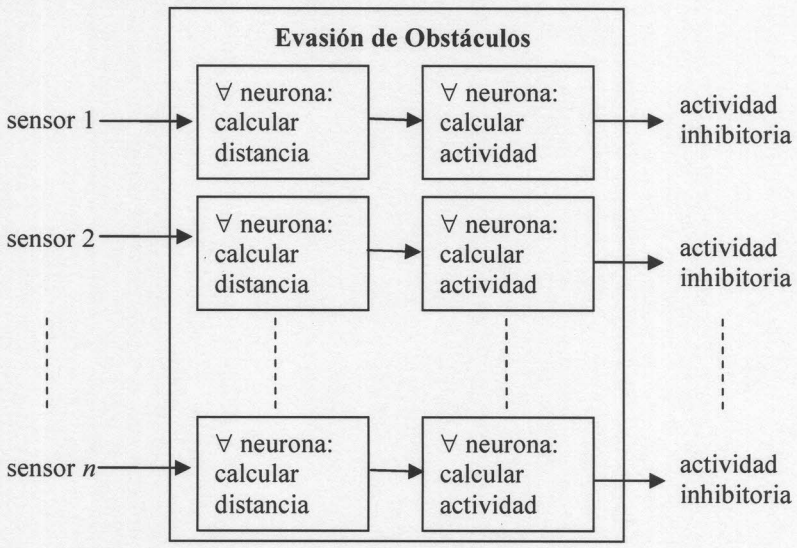


Figura 4.4.4: Gráfico de la función de actividad.

4.4.2 Evasión de obstáculos

El módulo de evasión de obstáculos utiliza el mismo EKM que para la localización de objetivo. El robot posee n sensores alrededor de su cuerpo de forma de detectar obstáculos: Cada sensor activo posee una dirección α_j fija y codifica una distancia d_j al obstáculo detectado. Los valores de α_j se encuentran determinados por la posición y dirección del sensor en el cuerpo del agente, por lo cual los valores de ángulo serán los mismos durante todo el procesamiento para cada uno de los sensores, solo variando el valor de distancia en función de la cercanía del robot con los obstáculos detectados.

Al igual que el modulo de alcance de objetivo la evasión de obstáculos realiza el cálculo de distancias sobre los valores ingresados como entrada y los desplazamientos asociados con las neuronas, para luego obtener una actividad.



Cálculo de distancias

Los valores de distancia a obstáculos serán medidos con valores reales entre 0 y 1. Esta distancia no es la distancia real existente entre el obstáculo y el robot sino que es

una medida que indica el riesgo de colisión del agente con el obstáculo. Si el valor de distancia es cero el robot colisionará, si es uno el robot no colisionará, y los valores intermedios denotan valores de riesgo medio.

El cálculo de las distancias está basado en zonas. Se definen dos parámetros, *distancia_choque y distancia_segura,* los cuales determinan los límites de estas regiones. Los valores de estos parámetros corresponden a distancias centradas en el obstáculo, medidas sobre la dirección perpendicular a la dirección del obstáculo respecto del robot. De esta manera, se determinan dos ángulos (considerando sólo un semiplano), que establecen los limites de tres regiones.

- La zona que hemos denominado libre de colisión es la región más alejada al obstáculo. Los valores de distancia dentro de esta zona tendrán el valor uno.

- La región más cercana al obstáculo se denomina zona de choque. Los valores de distancia en esta zona se encuentran fijos e iguales a cero. Esta zona se define como la región en la que la distancia real al obstáculo es menor o igual al parámetro *distancia_choque*.

- La región comprendida entre el agente y el punto ubicado a *distancia_choque* del obstáculo y dentro del triángulo formado por la *distancia_choque*, se denomina zona intermedia B. Sus valores de distancia crecen cuadráticamente desde cero hasta uno hacia la posición del robot en la dirección entre el obstáculo y el robot.

- La región establecida por los triángulos formados por la *distancia_segura* y *distancia_choque* se denomina zona intermedia A, y sus valores varían linealmente en la dirección perpendicular a la dirección entre el obstáculo y el robot, tomando como valor inicial el que posee en la zona de choque y creciendo a uno hacia la zona libre de colisión.

El siguiente gráfico muestra estas zonas y como los valores de distancia son afectados.

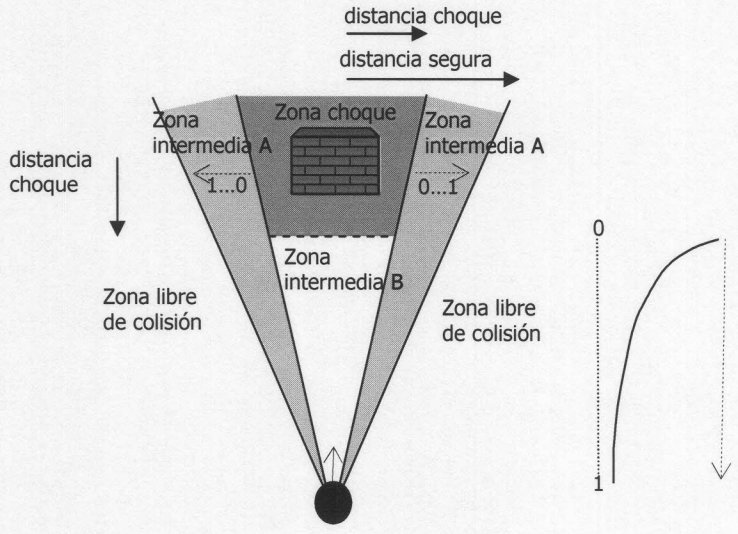


Figura 4.4.5: Delimitación de zonas para el cálculo de distancias.

Luego, la función de distancia utilizada para este módulo es calculada para cada sensor *j* de la siguiente forma:

1. Ingresar al sistema la posición detectada por el sensor j: $u_j = (\alpha_j, d_j)$

2. Dados los valores establecidos para los parámetros *distancia_choque* y *distancia_segura*, calcular para toda neurona *i* un valor de salida $d_{i,j}$ respecto del valor ingresado, utilizando el siguiente procedimiento:

// determinación de las zonas, en función de las pendientes.
pend_neurona ← | *tan(angulo(Posición(*w_i*),* u_j*))* |
pend_inical ← *distancia_choque* / ||u_j||
pend_final ← *distancia_segura* / ||u_j||
proyeccion_neurona ← (*Posición(*w_i*)* × u_j*)* / ||u_j||

// si el agente se aleja del obstáculo
// entonces se encuentra en la zona libre de colisión
Si *proyeccion_neurona* < *0*, ***entonces***
  $d_{i,j}$ ← *1*
Si no *// el robot se acerca al obstáculo*
  // si se está dentro de la zona de choque, la distancia es cero
  Si *Distancia(Posición(*w_i*),* u_j*)* < *distancia_choque*, ***entonces***
    $d_{i,j}$ ← *0*
  Si no *// no esta dentro de la zona de choque*
    // si está en la zona libre de colisión, la distancia es uno
    Si *pend_neurona* ≥ *pend_final*, ***entonces***
      $d_{i,j}$ ← *1*
    // si se encuentra en la zona intermedia B, la distancia es cero
    Si no, Si *pend_neurona* ≤ *pend_inicial*, ***entonces***
      $d_{i,j}$ ← *0*
    Si no
      // si se encuentra en la zona intermedia A, la distancia se calcula en forma lineal entre cero y uno
      $d_{i,j}$ ← *(pend_neurona – pend_inicial) / (pend_final – pend_inicial)*
    Fin si
    // para todas las zonas, salvo la de libre colisión, se procura hacer continua y suave la función de distancia
    Si $d_{i,j}$ < *1*, ***encontes***
      valor_proporcional ← *Minimo((||*u_j*|| - distancia_choque), proyeccion_neurona)) / (||*u_j*|| - distancia_choque)*
      valor_proporcional ← *(valor_proporcional)*^{2}
      $d_{i,j}$ ← *1- ((1 -* $d_{i,j}$*) valor_proporcional)*
    Fin si
  Fin si
Fin si

[PROC 4.4.1]

El algoritmo sigue una línea similar a la utilizada por los campos potenciales en la detección de obstáculos durante la navegación con robots móviles. Esta metodología genera un campo para el espacio de navegación del agente basado en funciones potenciales arbitrarias. La función tipo es similar a la de la fuerza electrostática o la de la ley de atracción gravitatoria, donde la fuerza potencial es inversamente proporcional al cuadrado de la distancia entre el robot y los objetos del entorno [HJRPP/05]. La misma idea fue aplicada en este trabajo: En las leyes físicas la fuerza electrostática varía de forma inversamente proporcional al cuadrado de la distancia a la carga eléctrica, y en el método propuesto se busco que el valor de salida de la neurona varíe proporcionalmente al cuadrado de la distancia al obstáculo. La diferencia entre hacerlo en forma inversa o directa surge del contraste entre que la fuerza eléctrica debe ser alta en cercanías de la carga eléctrica, y en cambio en esta ocasión la salida de la neurona tiene que ser baja en cercanías del obstáculo.

Cálculo de actividad de Obstáculos

Los vectores de pesos en las neuronas representan las posiciones que puede obtener el robot durante la ejecución de un control de motor. En el método propuesto en esta tesis los valores de actividad que devolverá cada neurona se encuentran centrados en la posición detectada del obstáculo, a diferencia de otros métodos [LKA/04] [LKA/03] que centran la función determinante de la actividad en el desplazamiento asociado con la neurona ganadora. Hacerlo de esta forma, en determinadas circunstancias, puede ocasionar que el robot interprete que no puede llegar al objetivo. En la figura siguiente, se puede observar ambas propuestas.

La figura 4.4.6 (a) se corresponde a la arquitectura desarrollada por Low, Khleng y Ang, donde se puede apreciar que la actividad de los obstáculos inhiben la actividad generada por el objetivo de tal forma que la neurona ganadora no es la más apropiada. En cambio la figura 4.4.6 (b), correspondiente a nuestra implementación, la neurona ganadora en el EKM no se ve afectada por la zona inhibida tras el cálculo de actividad de obstáculos, puesto que dicha actividad se encuentra centrada en el mismo obstáculo y no en las neuronas.

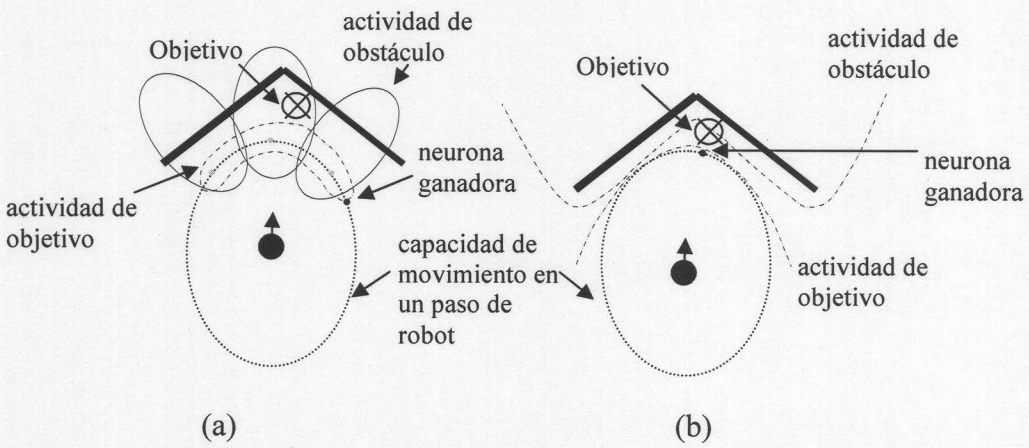


Figura 4.4.6: La figura (a) corresponde a la arquitectura desarrollada por Low, Khleng y Ang, la figura (b) se corresponde a nuestra implementación.

Al igual que en el cálculo de actividad del objetivo los valores de actividad se encuentran en el rango *[0, 1]*. Luego para obtener la actividad neuronal determinada por cada sensor *j* se realiza el siguiente procedimiento:

1. Por cada neurona *i* calcular el valor de distancia $d_{i,j}$ según el procedimiento [PROC 4.4.1], obteniéndose el vector de distancias D_j:

$$D_j = (d_{1,j}, d_{2,j}, \ldots, d_{i-1,j}, d_{n,j}) \;\; d_{i,j} \geq 0 \; \forall i$$

2. Obtener, el máximo valor:

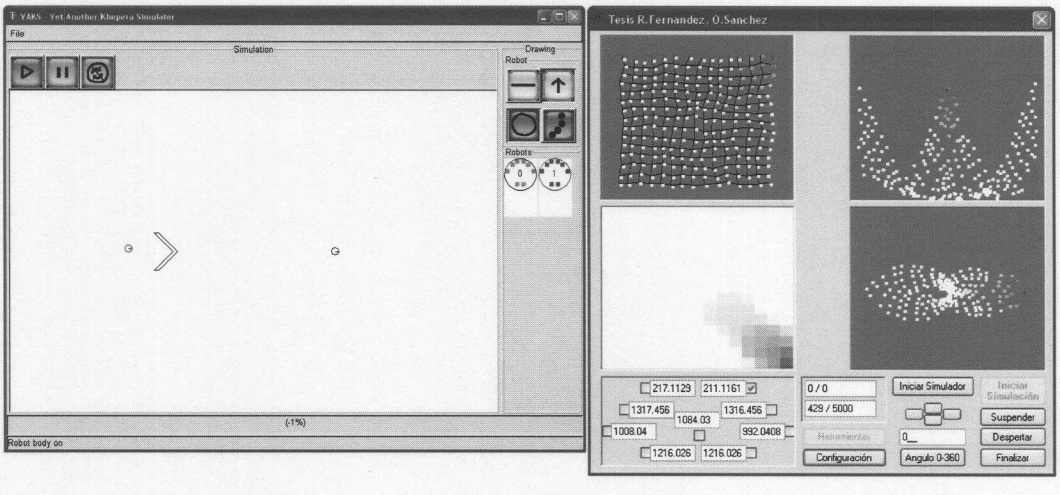
$$d_{\max} = \max(D_j)$$

3. Calcular la actividad B_j para cada neurona *i*:

$$B_{ij} = \begin{cases} 1 \quad si \quad d_{\max} = 0 \\ \dfrac{(-d_{i,j} + d_{\max})}{d_{\max}} \quad si \quad d_{\max} > 0 \end{cases}$$

Cuando un sensor detecta un obstáculo que está ubicado más allá de la distancia del objetivo, entonces el mismo no es tomado en cuenta para el cálculo de la actividad. Es decir los valores B_{ij} correspondientes a este sensor son iguales a cero.

Las siguientes pantallas fueron obtenidas del software que se ha desarrollado y el simulador utilizado. En las mismas se observa la actividad resultante de haber evaluado la red en la circunstancia que puede apreciarse en el escenario propuesto. Por razones de simplicidad sólo un sensor se encuentra activo. Seguidamente se muestra la pantalla de simulación para poder observar la posición del robot con respecto al obstáculo. Los colores más intensos representan valores de actividad más elevados.



(a) (b)

Figura 4.4.7: (a) escenario propuesto. (b) mapa autoorganizado de desplazamientos. Se representa los pesos de cada neurona en coordenadas cartesianas y polares.

Para mayor claridad se puede apreciar a continuación el gráfico de los valores de actividad obtenidos.

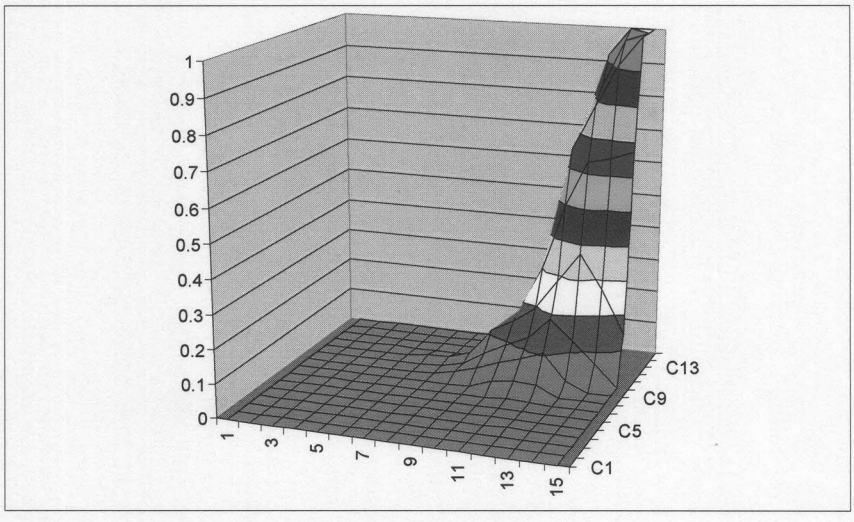


Figura 4.4.8: Gráfico de la función de actividad.

4.4.3 Integración de Actividades

Luego de realizar el cálculo de las actividades respecto del objetivo y de los obstáculos detectados por cada sensor, se realiza una integración de las actividades a fin de obtener como resultado la neurona cuya actividad final sea máxima. Esta neurona será la que posea el control de motor a ejecutarse y el desplazamiento asociado que será producido. Para eso, se sumarán los valores de actividades del objetivo y se restarán los valores de actividades de obstáculos, de forma tal de que la actividad de objetivo funcione como señal excitatoria y la de los obstáculos como señales inhibitorias.

La actividad resultante está limita al rango [-1, 1] y es determinada por el siguiente procedimiento, el cual es realizado para cada neurona *i*:

1. Obtener los valores de actividad A para el objetivo, y los valores de actividad B_j para cada sensor de obstáculos j, según los procedimientos anteriormente descriptos.

2. Obtener la cantidad de sensores de obstáculos q, para los que se ha percibido un obstáculo a menor distancia que la posición del objetivo.

3. Calcular la actividad F_i correspondiente a la neurona i, que resulta de aplicar el siguiente algoritmo, donde A_i corresponde a la actividad de objetivo de la neurona i, y B_{ji} corresponde a la actividad de obstáculos detectado por el sensor j para la neurona i:

$F_i \leftarrow A_i \times min(max(1, q), 2)$
Para cada *sensor j*
  $F_i \leftarrow F_i - B_{ij}$
Fin para
Si $F_i < 0$, ***entonces***

$F_i \leftarrow F_i / q$
Si no
$F_i \leftarrow F_i / min(max(1, q), 2)$
Fin Si

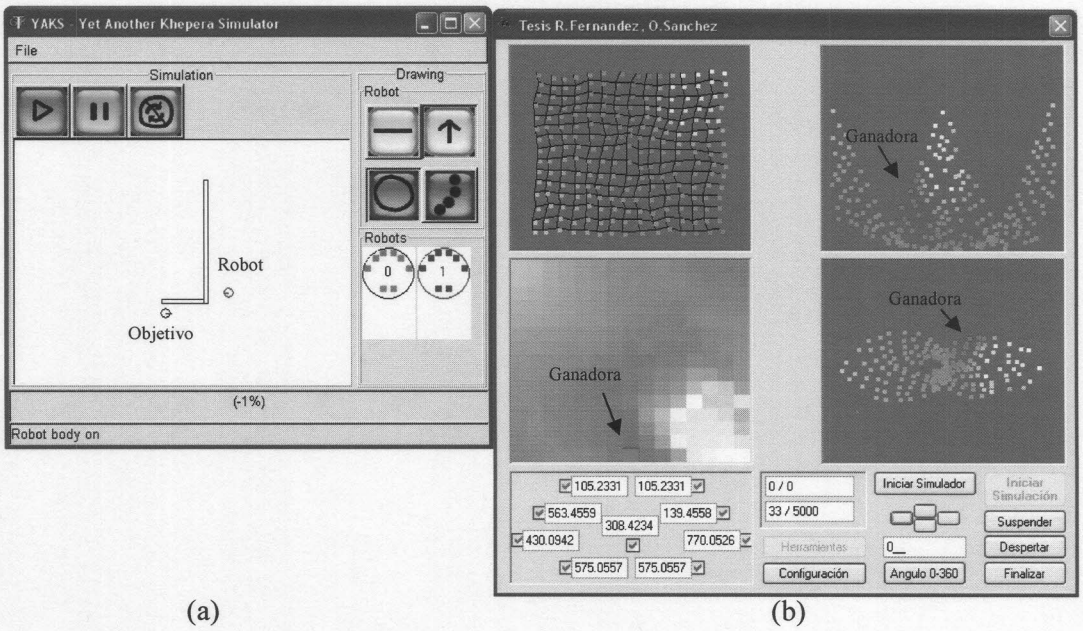
En la primera línea del procedimiento se potencia la actividad del objetivo, en la medida de la cantidad de sensores que pueden implicar superposiciones de actividades inhibitorias. En el caso particular del robot Khepera, en función de la disposición de sus sensores de obstáculos, y la delimitación de zonas indicada en la figura 4.4.5, resulta en un máximo potencial de *2* (*dos*) actividades inhibitorias superpuestas. Es por eso que la función mínimo se aplica a *2* y *max(1, q)*, donde *max(1, q)* implica al menos el valor *1 (uno)* cuando los sensores no perciben obstáculos

La actividad resultante determina entonces la neurona ganadora *s* como aquella que posee el máximo valor de actividad, de forma que se cumpla la siguiente condición:

$$F_s = max_{\,i=1...n} \{ F_i \}$$

Luego el control de motor ejecutado se corresponde al vector de pesos $Cg = (Cig, Cdg, tg)$, el cual producirá el desplazamiento $V_g = [d_g, \alpha_g, \theta_g]$.

En la figura 4.4.9 puede apreciarse cómo en un escenario en el cual el robot se encuentra frente a una pared, la neurona ganadora es aquella que produce un movimiento que logra evitar que el agente colisione con la pared y se dirija hacia el objetivo. Puede observarse además la actividad en el mapa autoorganizado y los desplazamientos de las neuronas según las coordenadas cartesianas y polares. Los tonos verdes indican valores de actividad mayores a cero, mientras que los tonos grises indican valores de actividad menores a cero. Los tonos más intensos corresponden a valores de mayor actividad absoluta.



(a) (b)

Figura 4.4.9: (a) escenario propuesto. (b) pantalla del software desarrollado, donde se puede observar los desplazamientos asociados con las neuronas y la actividad generada por el método propuesto.

Se puede apreciar a continuación el gráfico de los valores de actividad obtenidos.

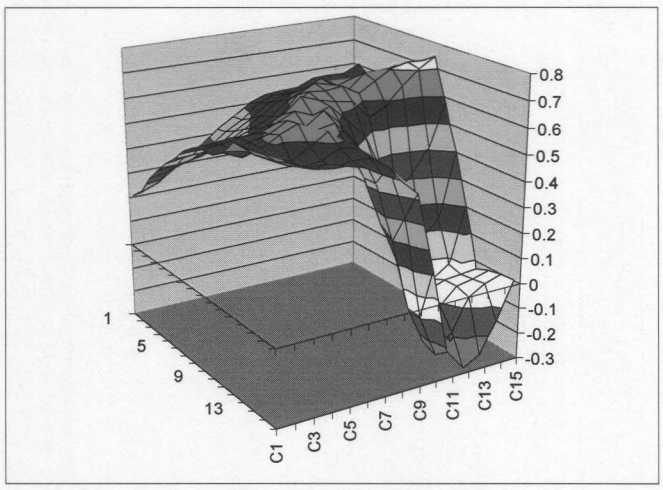


Figura 4.4.10: Gráfico de la función de actividad.

4.5 Situaciones Especiales

4.5.1 Objetivo

Como se ha explicado anteriormente, los parámetros β_α, β_θ le permiten al agente avanzar hacia el objetivo, acercándose con ángulo y orientación convenientes. Pero una vez que dicho agente consigue orientarse hacia su objetivo, no es necesario seguir priorizando dichas dimensiones, porque si aparece un obstáculo entre el agente y el objetivo, se le debe permitir tomar otro camino que no sea el de mejor ángulo y orientación hacia el objetivo, sino otro camino que le permita evitar satisfactoriamente el obstáculo.

Es por eso que el método propuesto no debe forzar a que permanentemente el agente se oriente hacia el objetivo. Se estableció una modalidad de actualización de los parámetros β_α, β_θ utilizados en el cálculo de las distancias durante la localización del objetivo, de forma de dar menos importancia a la orientación y al ángulo cuanto más orientado este el agente, para lo cual se utiliza la siguiente expresión:

Dada la posición del objetivo $u_t = (\alpha_t, d_t)$

Si $|\alpha_t| \geq \frac{\pi}{2}$, **entonces**

$$\beta_\alpha \leftarrow \beta^*_\alpha, \beta_\theta \leftarrow \beta^*_\theta$$

Si no

$$\beta_\alpha \leftarrow \beta^*_\alpha \left(\frac{\alpha_t}{\pi/2}\right)^2, \beta_\theta \leftarrow \beta^*_\theta \left(\frac{\alpha_t}{\pi/2}\right)^2$$

Fin Si

[PROC 4.5.1]

donde β^*_α, β^*_θ corresponden a parámetros constantes, establecidos originalmente en la configuración del método, y β_α, β_θ son parámetros dinámicos que varían en función del algoritmo descripto y son utilizados en el cálculo de la distancia.

Se buscó escalar los parámetros β de ángulo y orientación cuadráticamente, para el intervalo $\left[-\frac{\pi}{2},0\right]$ tomando valores desde el valor original hasta cero, y en el intervalo $\left[0,\frac{\pi}{2}\right]$ tomando valores desde cero hasta el valor original.

En las ocasiones en las cuales el agente se ubica muy cercano a un obstáculo estos valores también debes ser modificados, ya que el robot debe dejar de priorizar el llegar al objetivo respetando ángulos y orientaciones, de forma de evitar colisionar con el obstáculo, en cuyo caso se lleva a cabo el siguiente procedimiento que debe ser ejecutado a continuación del procedimiento [PROC 4.5.1]:

Dadas las distancias $\{d_{1..j}\}$ que cada sensor j detecta para con los obstáculos

$min\_distancia \leftarrow min(\{d_{1..j}\})$
Si min_distancia < *distancia_choque*, ***entonces***
$\beta_\alpha \leftarrow 0\ ,\ \beta_\theta \leftarrow 0$
Si no, Si *min_distancia* < *distancia_segura*, ***entonces***

$$\beta_\alpha \leftarrow \beta_\alpha \left(\frac{\min\_dis\tan cia - dis\tan cia\_choque}{dis\tan cia\_segura - dis\tan cia\_choque} \right)$$

$$\beta_\theta \leftarrow \beta_\theta \left(\frac{\min\_dis\tan cia - dis\tan cia\_choque}{dis\tan cia\_segura - dis\tan cia\_choque} \right)$$

Fin Si

[PROC 4.6.1.2]

La idea es escalar los parámetros β de ángulo y orientación linealmente entre cero y el valor original, en función de la distancia al obstáculo más cercano detectado, para distancias comprendidas entre cero y el valor del parámetro *distancia_segura*.

4.5.2 Obstáculos

Hay situaciones en las cuales el agente se encuentra estancado, es decir que intenta fallidamente una y otra vez evitar un obstáculo, por ejemplo ante la presencia de obstáculos grandes de tipo cóncavo, a los cuales el agente se ha acercado demasiado y no puede evitar en un solo paso. Ante estas situaciones se hace necesario entonces ejecutar algún mecanismo para que el agente evite dichos obstáculos. En el presente trabajo esto se logra aumentando el valor del parámetro *distancia_segura*, con el fin de crear una zona de exclusión mayor alrededor del obstáculo y así poder evitarlo.

Lo anterior se logra siguiendo los lineamientos que se pasan a detallar. Durante la ejecución son almacenados los m últimos valores de distancia $\{d_{1..,m}\}$ al objetivo, de forma de poder evaluar si el robot se encuentra en una situación en la cual está estancado, observando si todas la m distancias no varían de su media o promedio en más

de la máxima distancia que el robot puede recorrer en un solo paso. Con toda esta información se actualiza el parámetro *distancia_segura* con la siguiente expresión:

Si $d_1 > d_2 > \ldots > d_m$, ***entonces***
$distancia\_segura \leftarrow distancia\_segura^*$
Si no, Si $\forall i \mid d_i - Promedio(\{d_{1\ldots m}\}) \mid < recorrido\_maximo$, ***entonces***
$$distancia\_segura \leftarrow \left(\frac{distancia\_segura * 110}{100} \right)$$
Fin Si

donde el parámetro *recorrido_maximo* es aquel que mide la distancia máxima que puede desplazarse el agente en un solo paso. La constante $distancia\_segura^*$ corresponde al valor establecido originalmente en la configuración del método, y *distancia_segura* es el valor dinámico que varía en función del procedimiento descripto.

Capítulo 5

En esta sección se pretende dar un acercamiento a las características del robot Khepera, el simulador utilizado y las modificaciones que han realizado al mismo en la presente tesis, seguidamente se darán a conocer los detalle de implementación y utilización del software desarrollado que hemos desarrollado.

5.1 El robot Khepera

El robot Khepera fue diseñado originalmente por el grupo "K-Team" en Suiza como una herramienta para investigación y docencia. Este permite el desarrollo y la prueba de algoritmos como son el seguir trayectorias, evitar obstáculos o la recuperación de información visual. Existe una versión básica del robot a la cual se le puede agregar una serie de aditamentos como son un sensor de visión lineal, cámara de vídeo o un brazo de dos grados de libertad. Para controlar el robot, se tiene una interfaz de programación que consta de una serie de comandos básicos implementados en el entorno MATLAB. La figura 5.1.1 presenta un diagrama del robot Khepera en su versión básica.

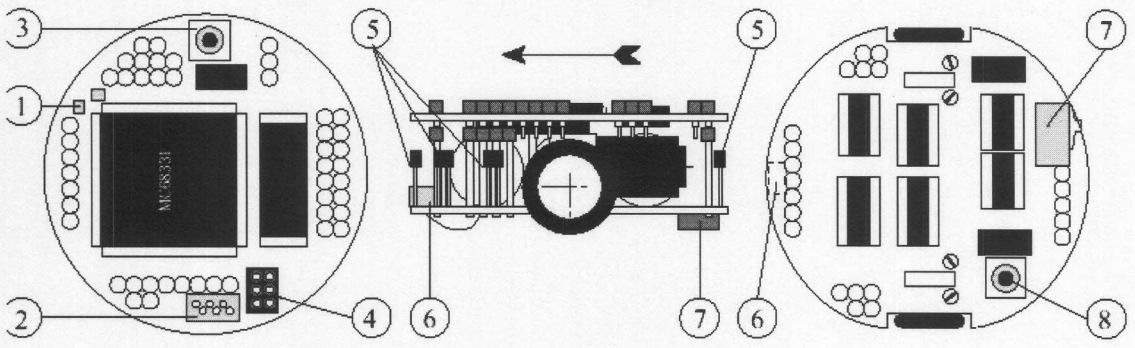


Figura 5.1.1: (1) LEDs, (2) Conector para comunicación serie, (3) Reset, (4) Jumpers para la selección del modo de operación, (5) Sensores de proximidad infrarrojos, (6) Conector de recarga de batería, (7) Interruptor de batería, (8) Reset.

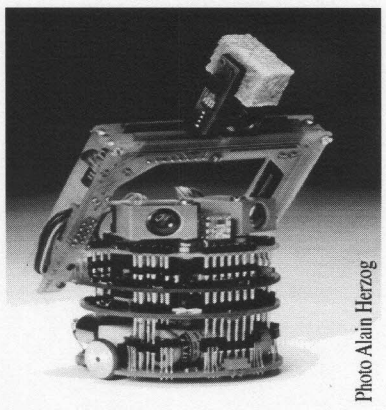


Figura 5.1.2: Foto del robot Khepera

El robot cuenta con una batería integrada, cuando el interruptor correspondiente está en posición de encendido, la batería provee de energía al motor. De otra forma es necesario contar con una fuente de alimentación externa.

Cabe mencionar que el robot Khepera es controlado por un microcontrolador. La memoria tipo ROM que tiene instalada, posee diferentes módulos con rutinas para el control del robot. Parte de estas rutinas incluidas en el BIOS (Sistema Básico de Entrada y Salida) se encargan de las funciones básicas como el control de los motores y la lectura de los decodificadores magnéticos. Otras rutinas se encargan del manejo del protocolo de comunicación con el usuario. Es posible configurar el robot de diferentes formas (control remoto, mandar y recibir datos, modo de prueba, etc). Se pueden elegir éstas con los conectores correspondientes al modo de operación.

5.1.2 Sensores de proximidad

El robot cuenta con ocho sensores de proximidad posicionados en el frente del mismo como se muestra en la figura 5.1.3. Estos sensores son de tipo infrarrojo, emisor y receptor, los cuales nos permiten dos tipos de mediciones:

Mediciones de luz visible: ésta se hace utilizando sólo la parte correspondiente al receptor, es decir el transmisor no emite luz. Se hacen mediciones cada 20 milisegundos.

Mediciones de la luz reflejada por obstáculos: aquí si se transmite luz infrarroja. De igual forma se hacen mediciones cada 20 milisegundos.

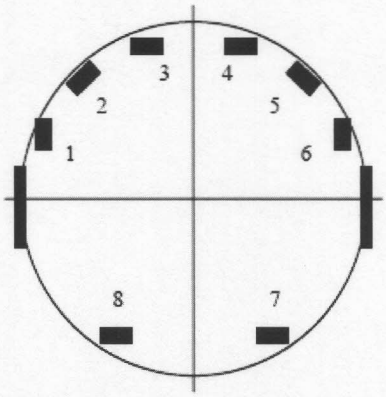


Figura 5.1.3: Distribución de sensores en el cuerpo del robot Khepera

La medición de los sensores depende de varios factores como pueden ser la distancia de la fuente emisora de luz, la intensidad de dicha fuente y el ángulo de incidencia. Por otro lado, las mediciones de luz reflejada dependen de las características del obstáculo (color y tipo de superficie).

Generalmente, este tipo de sensores proporcionan información en un rango de distancia relativamente corto. En el caso del robot Khepera estos sensores son útiles hasta un rango de 60 milímetros.

5.1.3 Control de los motores

Cada una de las ruedas del robot Khepera se mueve separadamente por medio de un motor de corriente directa con un sistema de reducción de 25:1. Cada motor posee un decodificador magnético acoplado a su eje. Cada decodificador tiene una resolución de 24 pulsos por revolución del motor lo que permite una resolución de 600 pulsos por

revolución de la rueda. Con esta simple estimación podemos hablar de que por cada milímetro de avance de las ruedas obtendremos 12 pulsos en los decodificadores.

El microcontrolador embarcado en el robot tiene control directo del voltaje de alimentación de los motores y de la lectura de los decodificadores magnéticos mediante rutinas de interrupción de bajo nivel. Existen tres modos de operación: control en posición del motor, control en velocidad y en control de pulsos en lazo abierto.

Ambos motores son controlados por un controlador PID (Proporcional Integral Derivativo) que opera como una rutina de interrupción del microcontrolador. El usuario puede ajustar cada parámetro de este controlador. Una vez ajustados estos parámetros, el control de los motores es transparente al usuario.

Por medio del comando adecuado se puede acceder a los parámetros de los perfiles de posición y velocidad. Los parámetros del controlador se pueden establecer por separado en cada caso.

Cuando se desea operar en el perfil de velocidad, la consigna de entrada será la velocidad deseada de la rueda, que generalmente se da en revoluciones por minuto. Se deben hacer otro tipo de consideraciones si se desea obtener la velocidad del robot. De la misma forma, al operar en el perfil de posición, está se refiere a una posición radial de la rueda y no a la posición del robot.

Como se mencionó en el párrafo anterior, el robot Khepera ofrece alternativas para operar en perfiles de posición y velocidad de cada uno de los motores. Sin embargo, para la experimentación en robots móviles se desea conocer la posición y/o velocidad del robot completo.

Un sistema de propulsión muy común en robots móviles consta de dos llantas controladas individualmente por motores separados. El robot Khepera tiene pequeños bordes en las partes frontal y trasera para asegurar la estabilidad. Esta configuración permite al robot girar sobre su propio eje, logrando una mejor movilidad en áreas congestionadas.

5.2 YAKS

YAKS (Yet Another Khepera Simulator) es un simulador de Khepera desarrollado por Johan Karlson [www.ida.his.se]. Este simulador es gratuito y de código abierto. El mismo ha sido modificado para permitir que sea controlado a través de otro programa independiente, utilizando los mismos comandos que el robot real. Posee las siguientes características:

- Permite incluir en el entorno obstáculos circulares, paredes, luces y definir zonas.
- Permite definir y manipular un número ilimitado de robots.
- Permite separar el programa de control del simulador, ya que los robots pueden ser manejados a través de una conexión TCP/IP.
- Soporta una gran variedad de sensores: proximidad, luminosidad, energía, encoders de las ruedas, compas y sensor de tierra (para detección de zonas).

El simulador YAKS es utilizado en la investigación y testeo de diversas aplicaciones en robótica, el mismo sin embargo no es del todo realista ya que no convive con restricciones propias del entorno y el robot utilizado: en el mundo real un robot debe convivir con fallas mecánicas, resbalamiento de las ruedas, agotamiento de la energía, fricción del piso o producidas por el ambiente como el viento, ruido en los sensores, entre otras.

5.2.1 Modificaciones realizadas al simulador

YAKS es un sistema de código abierto, por lo cual se pueden modificar sus funcionalidades o agregar otras a las ya existentes.

Los sensores de proximidad provistos en el sistema devuelven un valor real entre 0 y 1024 que representan la cercanía con un obstáculo. Esta magnitud originalmente es medida linealmente, sin embargo el algoritmo propuesto en esta tesis recibe como valor de los sensores una magnitud que indica la distancia y ángulo del robot con respecto al obstáculo detectado, por lo cual se estudió y modificó el código del simulador para lograr obtener los valores deseados.

Detectar un obstáculo linealmente no es del todo real, ya que los sensores no funcionan de esta forma. Se modificó entonces la detección del obstáculo, tomando como ángulo la orientación y posición del sensor implantado en el robot, asumiendo un área de detección angular que aumenta con la lejanía del robot.

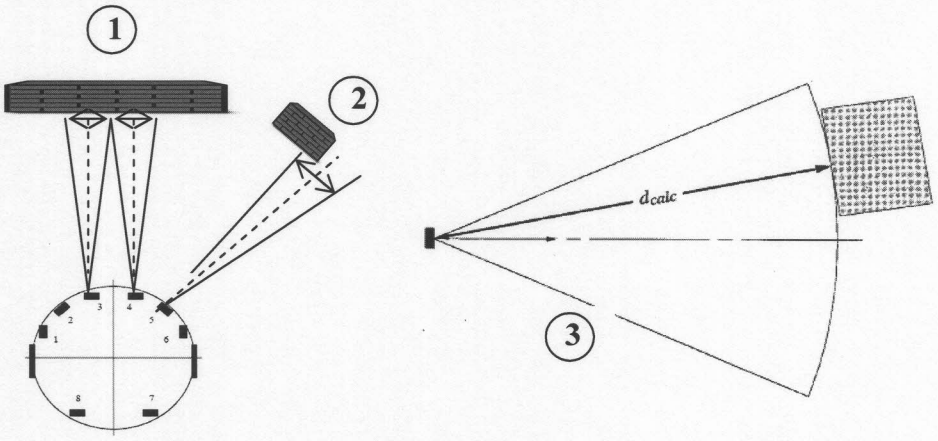


Figura 5.2.1: (1) se puede apreciar cómo la pared es detectada por ambos sensores, (2). la pared es detectada mediante la modificaciones realizadas. Notar que si se detecta la ubicación del obstáculo en forma lineal, este no es detectado por el sensor. Para mayor claridad se hace una ampliación de esta situación en la imagen (3), aquí se puede observar también cual es la distancia calculada que será utilizada como lectura de este sensor.

La funcionalidad adaptada también fue realizada para detectar otro tipo de obstáculos que poseen forma circular, en particular otro robot.

Los robots Khepera poseen sensores de muy corto alcance. Para realizar un mejor análisis del método que se propone, el simulador fue modificado para obtener un valor de distancia que no esté limitado.

5.3 Software desarrollado

Para poder estudiar con detenimiento el movimiento de un robot mediante el simulador YAKS, controlar al robot Khepera, evaluar el desempeño del mapa autoorganizado e investigar la locomoción del robot a su objetivo evitando obstáculos en el camino, se ha desarrollado un sistema por medio del cual se pueden acceder fácilmente a estas funcionalidades.

El sistema permite la configuración de las variables necesarias para el entrenamiento y la ejecución del robot en el simulador, por medio del mismo, se puede también suspender la ejecución para observar en detalle la red neuronal o la función de actividad neuronal propuesta para la identificación del objetivo y obstáculos, inclusive pueden importarse los valores a una planilla de cálculos.

En la figura 5.3.1 se puede observar la pantalla principal del sistema. Cabe aclarar que la lógica del método propuesta así como la red neuronal desarrollada forman parte del corazón de esta implementación.

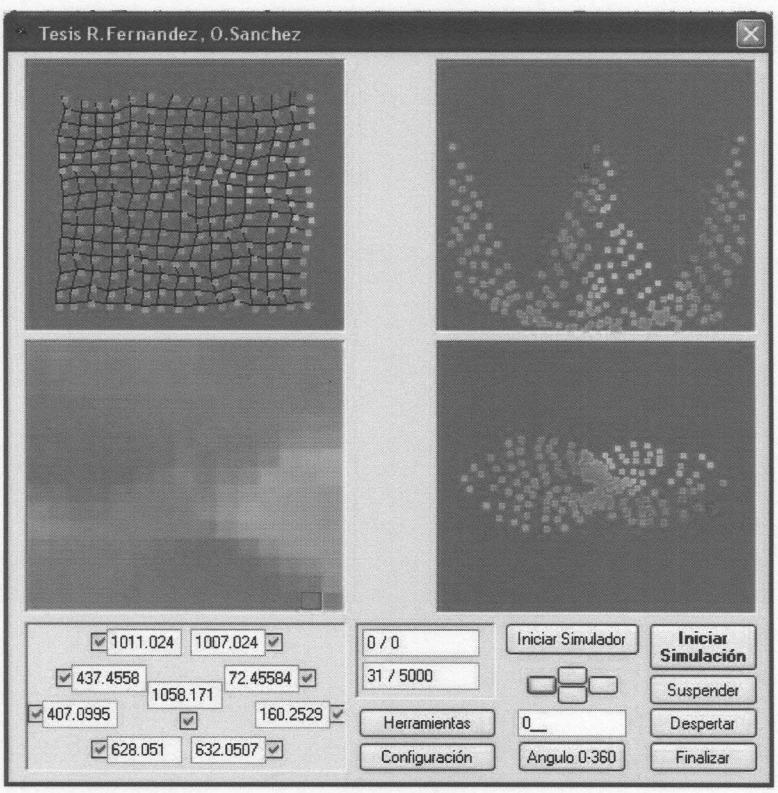


Figura 5.3.1: Pantalla del software que ha sido desarrollado.

El gráfico superior izquierdo de la pantalla corresponde con el mapa autoorganizado de entrada del EKM, el cual se corresponde con los controles de motor que puede ejecutar el robot. Los gráficos de la derecha identifican la capa de salida del EKM, correspondientes a los desplazamientos aprendidos por el agente, visto según la posición en coordenadas polares (arriba) y en coordenadas cartesianas (abajo). La actividad neuronal se ve reflejada en la subpantalla inferior izquierda. El color verde indica mayor actividad neuronal decayendo el tono de verde si la actividad para con el objetivo decrece y cambiando de blanco a negro en presencia de obstáculos.

La información debajo de la actividad neuronal se corresponde con la distancia que esta obteniendo cada sensor del robot ubicado en la posición que se refleja en la pantalla, el centro se corresponde con la distancia que indica la lejanía con el objetivo.

El sistema también contiene los comandos necesarios para poder maniobrar el robot si se desea girarlo o suspender su marcha, como así también acceder a las variables de configuración, pudiendo modificarlas en forma on-line y ver reflejado inmediatamente como este cambio afecta el movimiento del agente.

Capítulo 6

Este capítulo presenta un análisis del método propuesto y de la utilización de los parámetros correspondientes, como así también las formas de entrenamiento de la red neuronal.

6.1 Análisis y Resultados

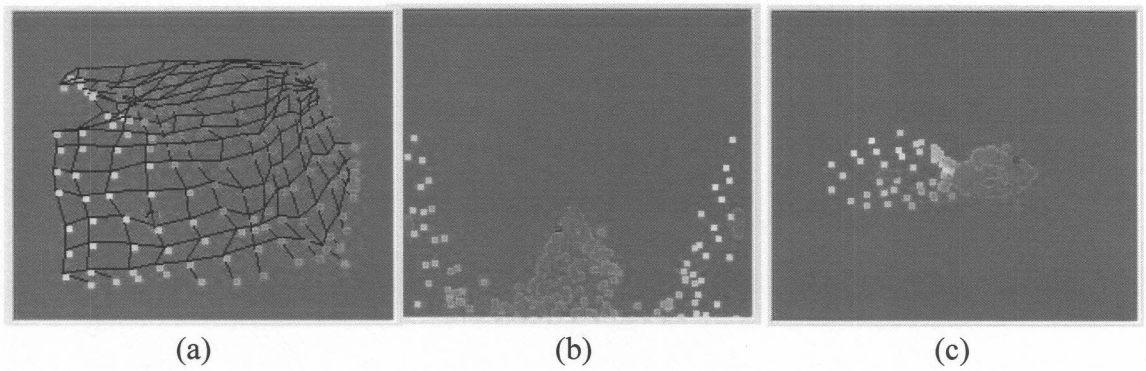
6.1.1 Entrenamiento

La observación y análisis del controlador de motor desarrollado fue realizado sobre un entrenamiento producido con antelación a la práctica de las pruebas realizadas, es decir la red se entrenó con anterioridad utilizando el simulador YAKS en un entorno libre de obstáculos. El robot ejecutó movimientos al azar y se observaron los desplazamientos producidos mediante los controles de motor que se ejecutaron.

Se analizó la posibilidad de realizar un entrenamiento mientras se lleva a cabo la tarea de llegar al objetivo como se menciona en los trabajos de Low, Khleng y Ang [LKA/04] [LKA/03], pero se decidió no utilizar este tipo de entrenamiento ya que para conseguir que la red se estabilice es necesario que sean ejecutados una gran cantidad de ejemplos, además de contener los inconvenientes planteados en los capítulos anteriores.

Luego de evaluar el entrenamiento producido utilizando distintas constantes para el algoritmo de aprendizaje, se determinó que realizando el entrenamiento con antelación es necesario una mínima cantidad de pasos para que la red converja, con lo cual puede realizarse una primera fase de entrenamiento y luego seguir entrenando la red mientras se alcanza al objetivo o directamente utilizar la red entrenada para lograrlo.

Otra de las alternativas es entrenar la red mientras se busca alcanzar el objetivo utilizando durante este proceso la técnica de exploración, con lo cual cada cierta cantidad de pasos se ejecutan controles de motor al azar de forma de que la red no colapse replicando la distribución acotada de los datos de entrada, en los siguientes gráficos se puede apreciar el mapa autoorganizado de controles de motor figura 6.1.1 (a) y el mapa autoorganizado de los desplazamientos figura 6.1.1 (b, c) obtenido luego de ejecutar 2000 pasos durante los cuales, cada 5 pasos se ejecutaron 2 pasos con controles de motor obtenidos al azar.



(a) (b) (c)

Figura 6.1.1: (a) capa de entrada del EKM, correspondiente a controles de motor, (b) capa de salida correspondiente a los desplazamientos, en coordenadas polares, (c) en coordenadas cartesianas.

Se puede apreciar cómo el haber utilizado un entrenamiento on-line, provoca que la distribución de las neuronas se corresponda con los desplazamientos mayormente ejecutados, es decir, con los desplazamientos que llevan al robot hacia delante. Por consiguiente existe una gran cantidad de neuronas que ejecutarán el mismo control de motor ya que como se aprecia en las figuras se encuentran solapadas unas a otras.

Para poder obtener una distribución de entrada que se corresponda con la mayor cantidad de desplazamientos posibles que puede realizar el robot y, como consecuencia, obtener un mayor rango de acción, es necesario un entrenamiento que sea llevado a cabo seleccionando la totalidad de los controles de motor al azar.

La red de Kohonen que ha sido utilizada para la realización de las pruebas es de 15 x 15 neuronas. Tomando en cuenta que cada rueda puede variar las velocidades en el rango [-10, 10], con lo cual se obtienen 441 posibles controles de motor diferentes, utilizar un mapa autoorganizado de 225 neuronas es una reducción del espacio de entrada considerable.

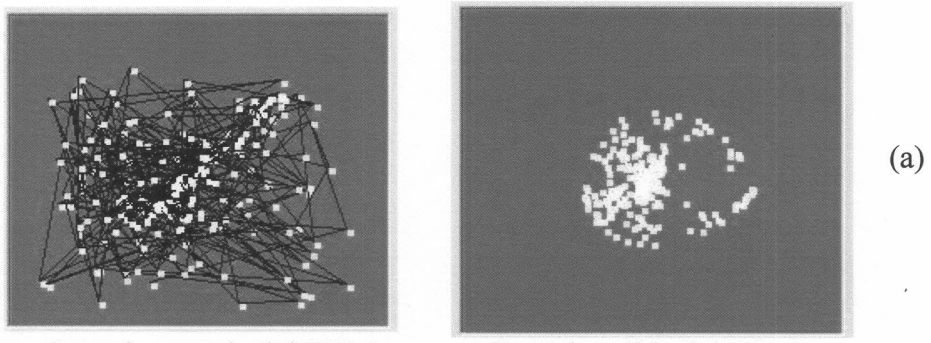
La velocidad de aprendizaje inicial se estableció en $\eta_{ini} = 0.5$ y su valor final fue ubicado en $\eta_{fin} = 10^{-5}$, el vecindario inicial se estableció $K = 15$, y vecindario final con valor $K = 0$. Estos valores fueron utilizados tanto en la capa de entrada que contiene los controles de motor como en la capa de salida que posee los desplazamientos a producir por los controles de motor asociados.

En la figura 6.1.2 puede verse la red neuronal al comenzar el aprendizaje. La misma fue inicializada con valores elegidos al azar. Seguidamente se observa los mapas del EKM luego de haber ejecutado 10.000 pasos, cada uno de 2 segundos de duración.

Como puede apreciarse en la figura 6.1.2 (b), la red ha resultado ordenada topológicamente. De no seleccionar correctamente los valores para la velocidad de aprendizaje y el vecindario inicial necesarios para el algoritmo la red puede no conseguir ordenarse correctamente, con lo cual no se cumple la premisa de que neuronas cercanas en el mapa autoorganizado contendrán vectores de pesos similares.

A diferencia de nuestra propuesta, en los papers de Low, Khleng y Ang [LKA/04] [LKA/03] la velocidad de aprendizaje y el vecindario son establecidos de forma de afectar solo a un pequeño porcentaje de neuronas alrededor de la ganadora y en escasa magnitud. Esta forma de aprendizaje es utilizada ya que el entrenamiento es realizado mientras el robot se dirige hacia el objetivo, con lo cual de esta forma se evita que la red colapse como se ha indicado en capítulos anteriores. También en este método, como se puede apreciar en el grafico de la figura 6.1.3 (a), existe una gran cantidad de neuronas con distancia cero. Esto se debe a que no han modificado sus vectores de pesos durante el aprendizaje, con lo cual estas neuronas pueden no representar efectivamente el desplazamiento que indican si el control de motor asociado fuese ejecutado.

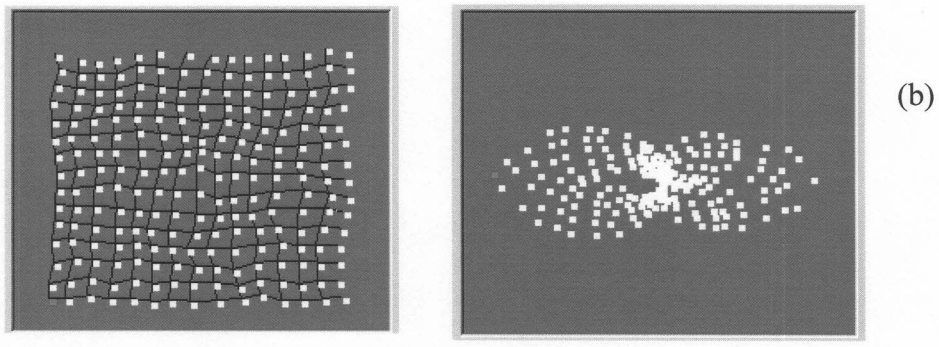
Se puede deducir de lo mencionado, que en el trabajo de Low, Khleng y Ang [LKA/04] no se produce un ordenamiento de las neuronas, incumpliendo la premisa de que neuronas cercanas en el mapa autoorganizado contienen vectores de pesos similares. Este resultado no se corresponde con nuestro trabajo, donde como puede apreciarse en la figura 6.1.3 (b), correspondiente a la capa de salida del EKM, los desplazamientos se encuentran ordenados y se corresponden con los controles de motor de la capa de entrada, los cuales también se encuentran ordenados.



Capa de entrada del EKM (controles de motor)

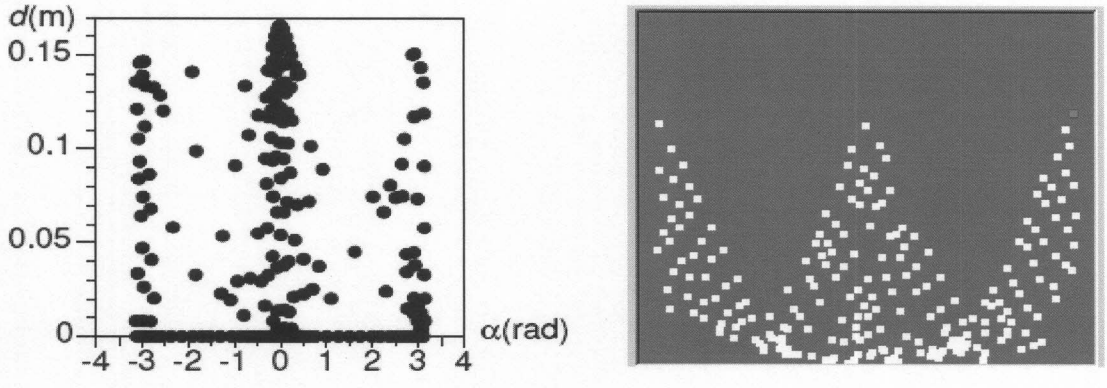
Capa de salida del EKM (desplazamientos)

(a)



(b)

Figura 6.1.2: (a) red neuronal al comienzo del entrenamiento, (b) red neuronal entrenada luego de la ejecución de 10.000 pasos.



(a) (b)

Figura 6.1.3: (a) Mapa autoorganizado de desplazamientos obtenido mediante el método de Low, Khleng y Ang [LKA/04] [LKA/03], (b) Capa de salida del EKM, correspondiente a los desplazamientos obtenidos mediante el método propuesto en esta tesis. Las escalas de distancia no son las mismas.

6.1.2 Alcance del Objetivo

Para analizar el método que se propone se utilizó el simulador YAKS y el software desarrollado. Las pruebas fueron realizadas sobre la red ya entrada, procedimiento que ha sido realizado utilizando 10.000 ejemplos obtenidos al azar.

Se expondrán gráficos sobre distintos escenarios, donde se detallan la cantidad de pasos necesarios para que el agente alcance un objetivo, partiendo desde distintas posiciones, de esta forma se pretende analizar el movimiento del agente.

Para poder determinar que el controlador de motor implementado carece de movimientos bruscos y que el agente se dirige al objetivo de forma conveniente, se particionó al ambiente en una grilla de 50 x 50 posiciones, centrando al objetivo en el centro del entorno, y haciendo que el robot se ubique en cada una de las posiciones de la grilla, orientándolo hacia el objetivo, para que se dirija desde cada una de estas posiciones hacia el mismo.

Luego de realizar el procedimiento descripto se obtuvo un gráfico que representa la cantidad de pasos que necesitó el robot desde cada posición de inicio para alcanzar el objetivo, este gráfico puede observarse en la figura 6.1.4.

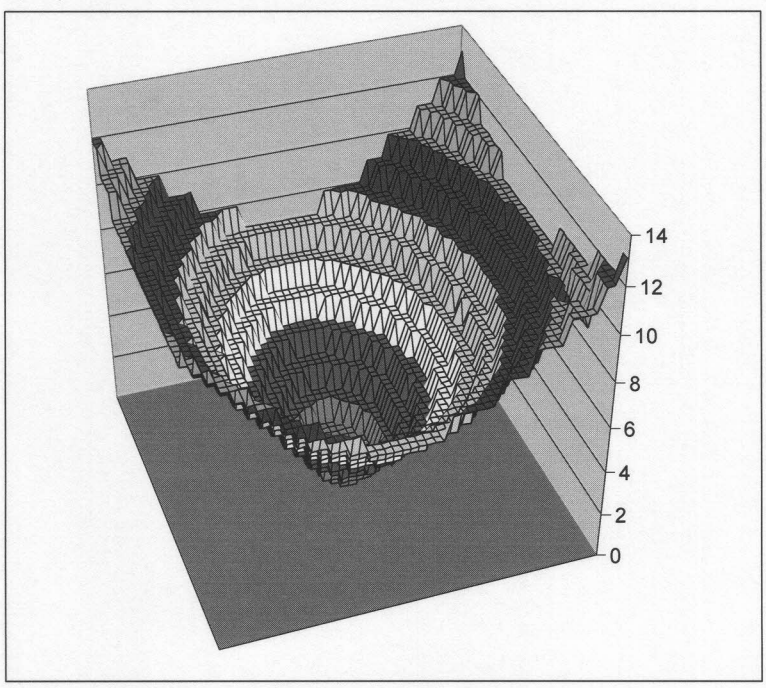
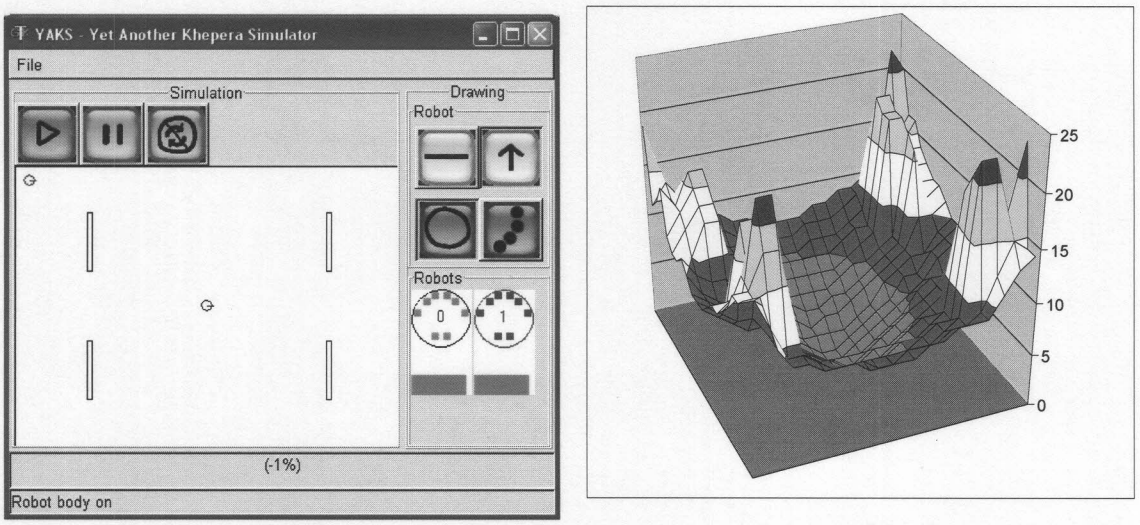


Figura 6.1.4: Cantidad de pasos ejecutados en función de la distancia al objetivo.

Como puede observarse la cantidad de pasos necesarios disminuyó con la cercanía al objetivo obteniéndose desplazamientos uniformes. El mismo procedimiento fue realizado en un entorno con obstáculos, para lo cual se incluyeron paredes en el mismo.

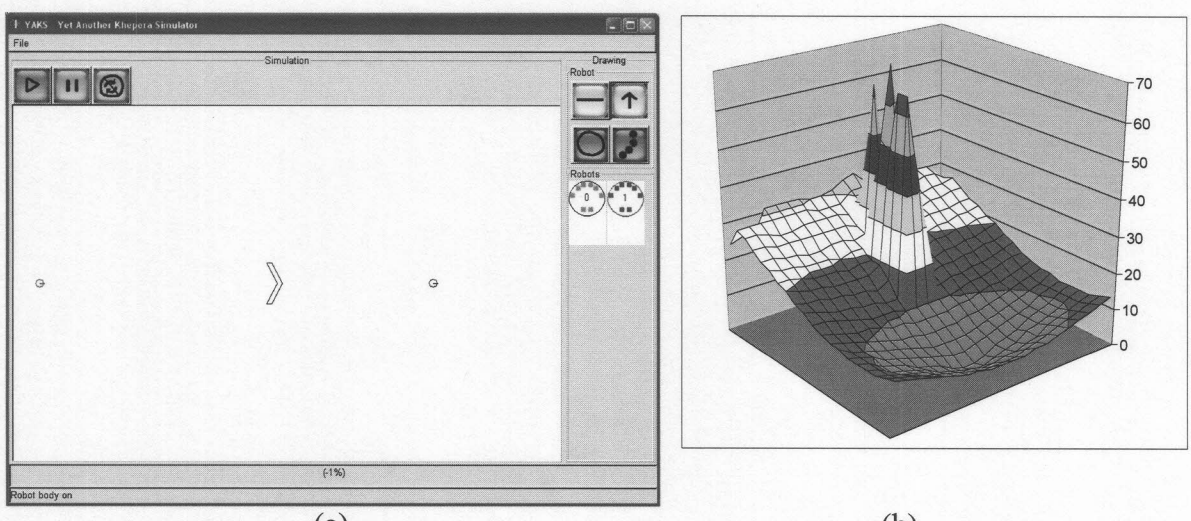


(a) (b)

Figura 6.1.5: (a) Escenario para el cual se ejecutó el experimento, (b) Cantidad de pasos ejecutados

La pantalla (a) muestra el escenario utilizado para la realización del experimento, (b) se corresponde con el gráfico obtenido. Puede apreciarse cómo la cantidad de pasos necesarios para llegar al objetivo aumenta cuando el agente se posiciona por detrás de las paredes, aunque de todas formas sigue siendo uniforme la cantidad de pasos.

Analizar un obstáculo cóncavo es también de gran utilidad para observar el comportamiento del algoritmo propuesto, ya que cuando el robot se encuentre dentro de la concavidad del mismo necesitara mayor cantidad de pasos para lograr salir de esta situación.



(a) (b)

Figura 6.1.6: (a) Escenario propuesto, (b) Cantidad de pasos ejecutados

Como puede apreciarse en los gráficos el método propuesto consigue, inclusive ante obstáculos cóncavos, llegar al objetivo de forma suave y precisa. Para continuar el

análisis, las pruebas deben centrarse en las variables utilizadas por el algoritmo y como éstas modifican el comportamiento del robot móvil en el entorno.

6.1.3 Parámetros de configuración

Para determinar el comportamiento de controlador de motor en base a los parámetros de sensibilidad de distancia, ángulo y orientación utilizadas, se preparó un escenario para el cual el robot debe sortear un obstáculo cóncavo y llegar al objetivo, el primer caso observado en la figura 6.1.7 corresponde a haber dado más importancia a la distancia que al ángulo y orientación del robot para con el objetivo.

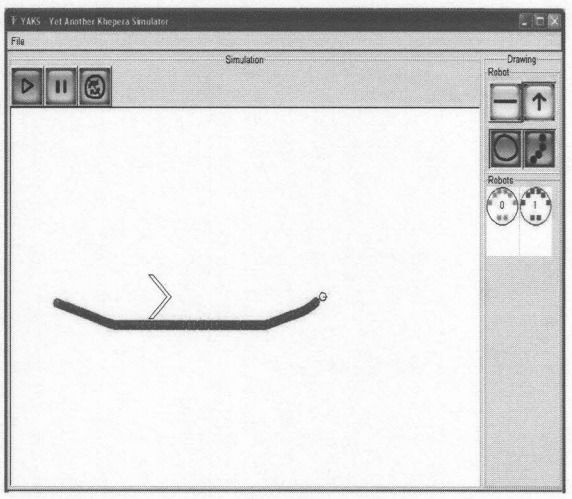


Figura 6.1.7: Trayectoria del robot ante un obstáculo cóncavo, tras dar mayor importancia al distancia.

Puede observarse cómo el robot se dirigió hacia el objetivo de forma precisa inmediatamente después de haber detectado el obstáculo. Si se le asignara mayor prioridad al ángulo el robot se comportaría de forma semejante pero en cada paso intentaría mejorar su posición con respecto al objetivo priorizando el ángulo que posee para con el mismo, en la figura 6.1.8 puede observarse este comportamiento.

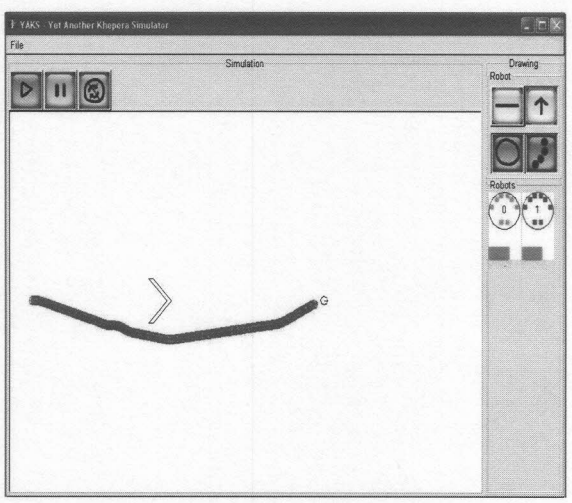


Figura 6.1.8: Trayectoria del robot ante un obstáculo cóncavo, tras dar mayor importancia al ángulo.

En este caso, el robot inmediatamente después de evitar el obstáculo se perfilo hacia el objetivo de forma de mejorar su ubicación priorizando el ángulo ante la distancia. Si la constante de orientación es la que se utilizará para determinar el comportamiento del agente, el comportamiento cambia notablemente. El mismo se detalla en la figura 6.1.9

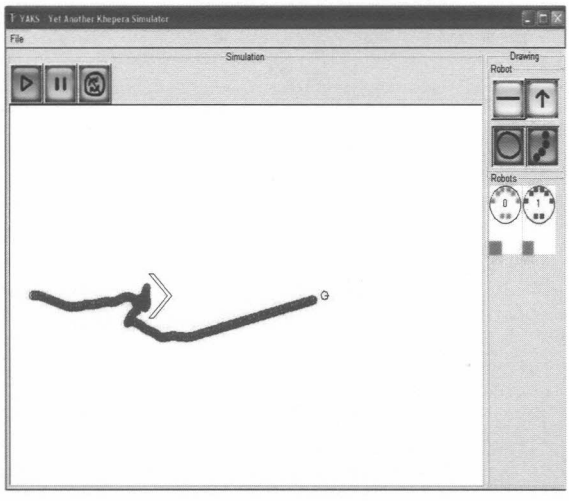


Figura 6.1.9: Trayectoria del robot ante un obstáculo cóncavo, tras dar mayor importancia a la orientación.

Esto sucede debido a que el agente intenta en cada paso conseguir la mejor orientación para con el objetivo, incluso dándole más importancia que acercarse al mismo u obtener una posición que se considere mejor a la del paso anterior, con lo cual queda encerrado dentro de la concavidad del obstáculo, al poder salir del mismo se puede observar como inmediatamente se dirige orientado perfectamente hacia el objetivo. En estos tres casos se determina con claridad que estas constantes deben ser establecidas convenientemente conforme a como se desea que el robot se comporte en el escenario.

Otra forma de modificar el comportamiento del robot es por medio del parámetro *distancia_segura*. El valor de este parámetro puede ser modificado ya sea por decisión de cómo se va a utilizar el agente en el entorno o porque durante la ejecución esta variable fue modificada de acuerdo a como se detalla en el apartado para los casos especiales. A continuación se puede observar como en un escenario determinado, variar el valor de la misma produce que el robot decida como evitar el obstáculo ejecutando controles de motor diferentes.

En la figura 6.1.11: (a) el coeficiente de distancia segura fue establecido en un valor de 1.5. Esto provoca que el agente considere una zona de exclusión pequeña alrededor del obstáculo y por lo tanto la neurona ganadora es aquella que provoca que el agente cambie su trayectoria levemente. En el gráfico (b) el coeficiente de distancia segura fue establecido en un valor de 5. Puede apreciarse como la neurona ganadora es aquella que provoca un cambio de trayectoria mayor al provocado en (a), en cambio para el caso de la figura 6.1.12 (c) el coeficiente fue establecido en 12, la distancia segura se ha incrementado notablemente, de forma de que la neurona ganadora es aquella que cambia la trayectoria del robot en una forma mayor al caso (c). En todos los casos el coeficiente de distancia de choque fue fijado en 0.85.

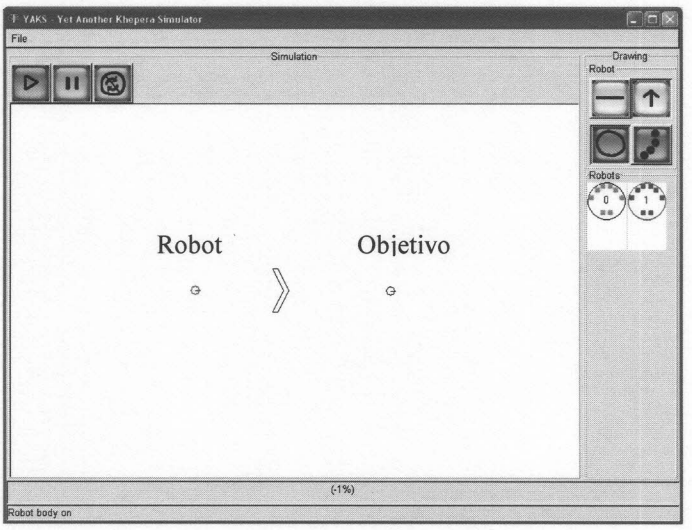
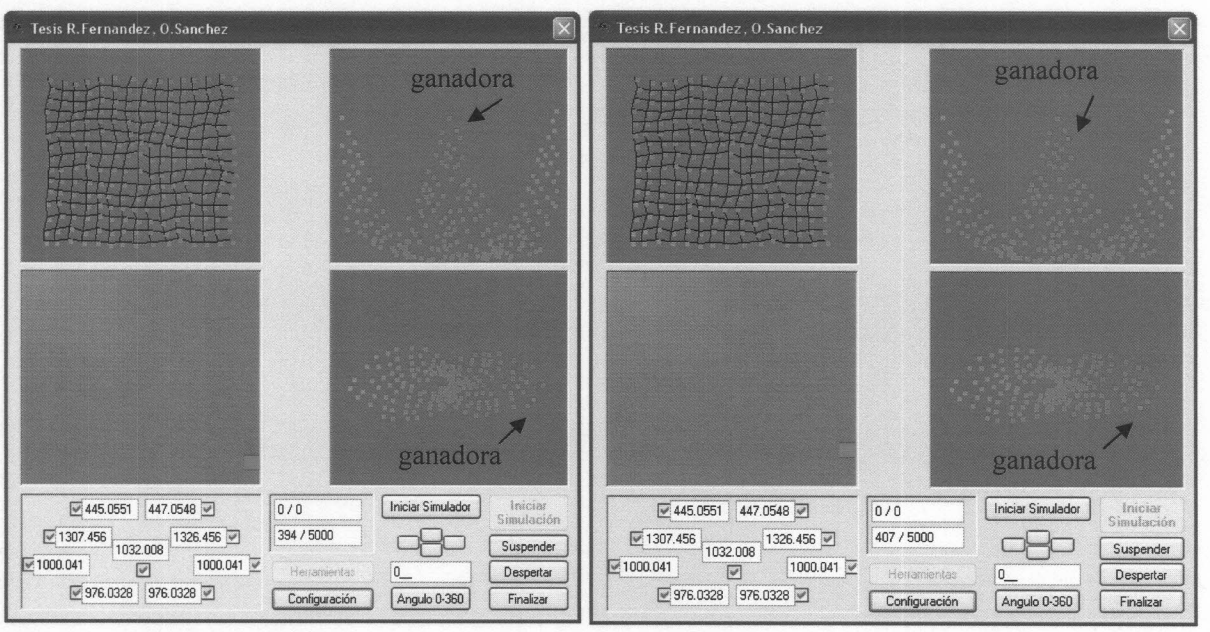
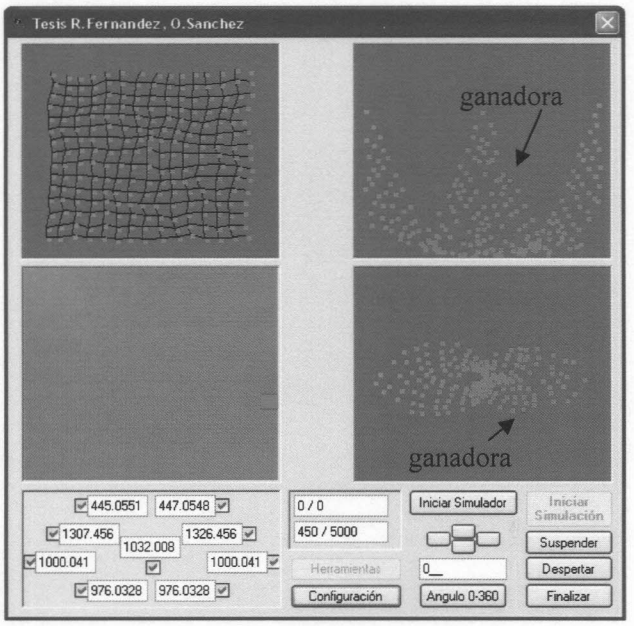


Figura 6.1.10: Escenario establecido en el simulador



(a) (b)

Figura 6.1.11: (a) Pantalla del software desarrollado, en la cual se aprecia la neurona, la actividad neuronal obtenida y las capas de entrada y salida del EKM. (b) Pantalla del software desarrollado y neurona ganadora obtenida.



(c)

Figura 6.1.12: (c) Pantalla del software desarrollado indicando la neurona ganadora obtenida.

Cabe destacar que la *distancia_segura* y *distancia_choque* se obtienen de multiplicar los respectivos coeficientes por el diámetro del robot, con lo cual los valores establecidos por medio del software desarrollado no indican directamente los valores de *distancia_segura* y *distancia_choque*, sino que se obtienen indirectamente a través de los coeficientes mencionados.

6.1.4 Situaciones complejas

El método presentado en este trabajo consigue resolver situaciones relativamente complejas, como por ejemplo, moverse en entornos estrechos como un laberinto, seguimiento de paredes y evitar obstáculos cóncavos, entre otros. Para demostrar este comportamiento, a continuación se ejemplifican escenarios en los cuales el robot debió sortear estas situaciones.

Un problema importante en la utilización de robots móviles es el movimiento del agente dentro de un laberinto. En el siguiente ejemplo se ha establecido un escenario con estas características, en el mismo existen dos puntos de control, que han sido agregados manualmente y a los cuales el agente debe dirigirse para poder llegar hasta el otro lado del escenario. Si bien el método propuesto no logra sortear todo el laberinto sin la existencia de estos puntos, de todos modos el robot puede avanzar hacia dichos puntos de control evitando colisionar con las paredes en este entorno de espacios reducidos. Cabe notar que los puntos de control pueden obtenerse mediante algún método de alto nivel, como por ejemplo a través de planeamiento.

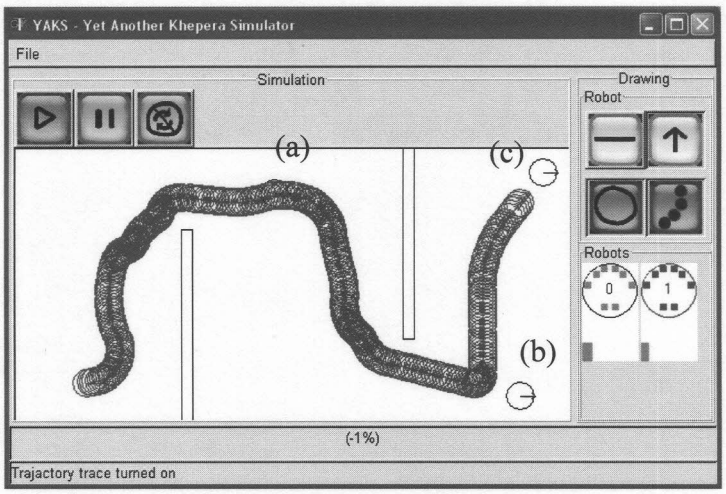


Figura 6.1.13: Trayectoria del agente dentro de un laberinto. (a) Primer punto de control, (b) segundo punto de control, (c) objetivo.

Otra situación que nuestro método logra resolver es el seguimiento de paredes, para observar este comportamiento se ha propuesto un escenario donde el objetivo fue establecido a la vuelta del final de la pared de forma de que el agente deba costear la misma para poder alcanzarlo.

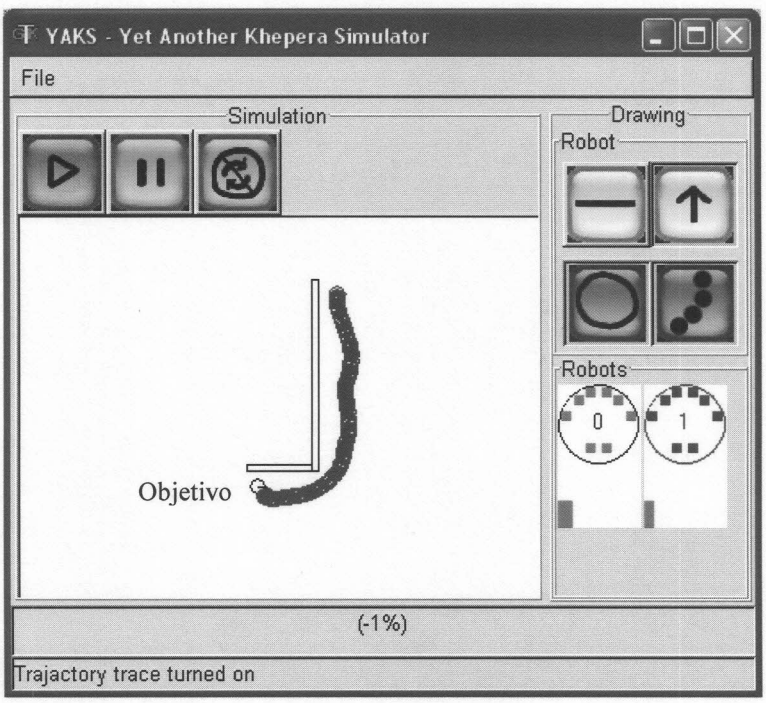


Figura 6.1.14: Trayectoria del robot costeando una pared

De forma de poder observar el movimiento del agente ante la presencia de un obstáculo cóncavo, se ha propuesto un escenario de estas características. El robot logra evitar colisionar con este obstáculo sin perjudicar de forma brusca su trayectoria, logrando sortearlo y llegar hacia el objetivo.

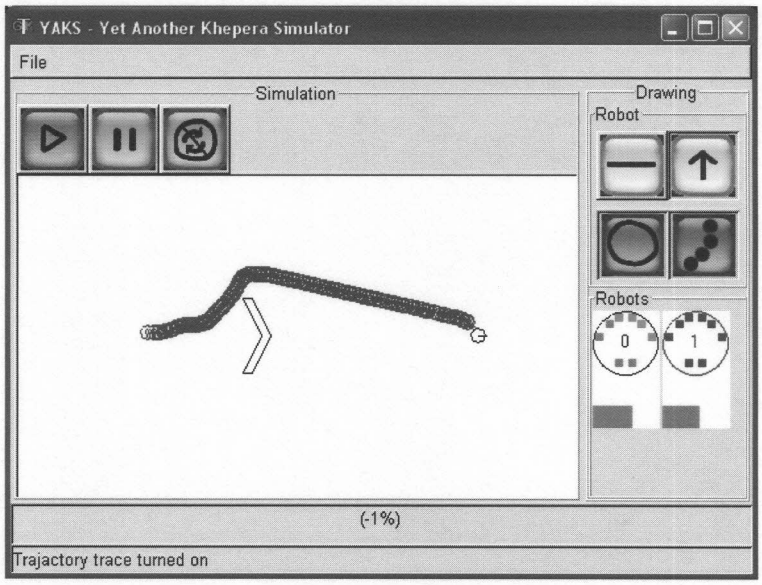


Figura 6.1.15: Trayectoria del robot tras evitar un obstáculo cóncavo

Luego de apreciar el comportamiento del agente dentro de estos escenarios, se lo ha situado ante escenarios más complejos. A continuación puede observarse un escenario donde existen un gran número de obstáculos. Puede observarse cómo el robot consigue una trayectoria sumamente aceptable aún sin la utilización de una técnica de planeamiento.

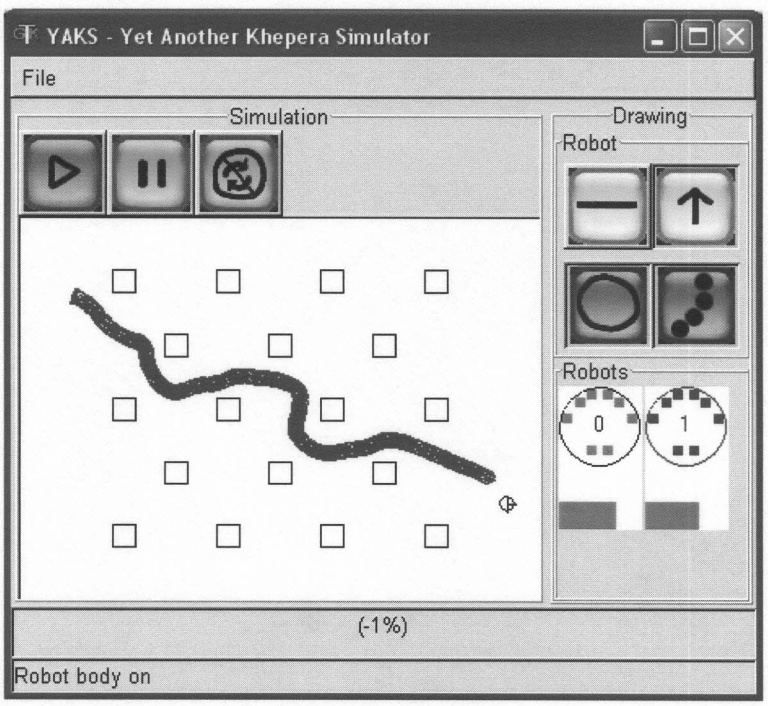


Figura 6.1.16: Trayectoria del robot en un escenario con gran cantidad de obstáculos

A continuación se propone otro escenario interesante donde puede apreciarse el comportamiento del agente.

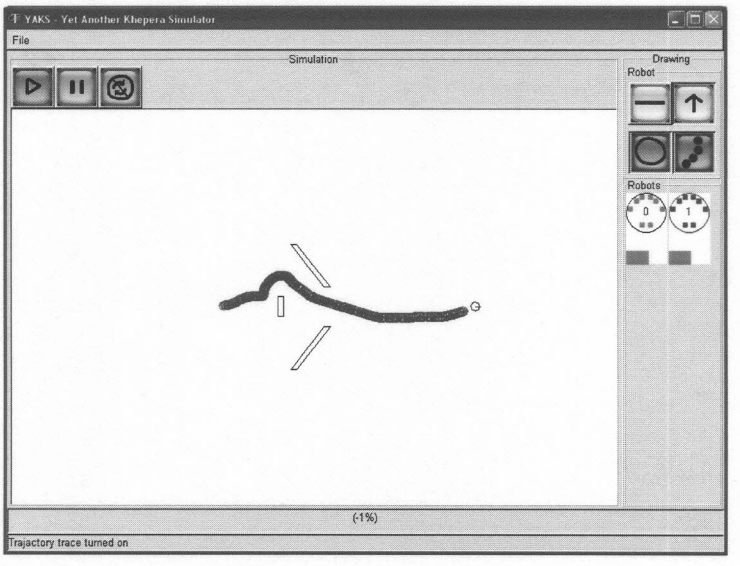


Figura 6.1.17: Trayectoria del robot en un escenario complejo.

Se ha descripto en varias oportunidades que nuestro método funciona de tal manera que el robot siempre avanza de frente. Recordar que la mayor cantidad de sensores se ubican en el frente del agente y esto hace que perciba mejor su entorno. Para mostrar este comportamiento se situó al objetivo detrás del robot. Puede observarse en la figura siguiente los pasos ejecutados por el agente y como ha ido cambiando la orientación en cada uno de ellos, orientándose en primer lugar hacia el objetivo y luego avanzando hacia el mismo.

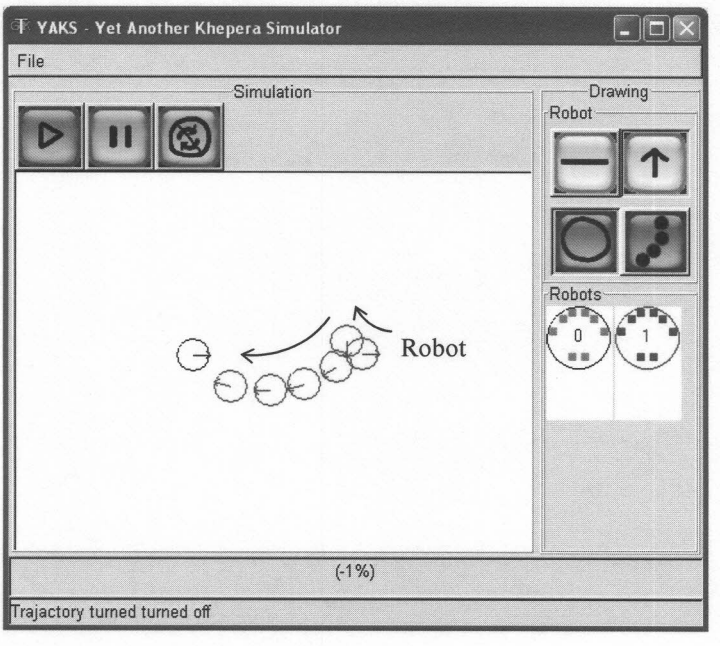


Figura 6.1.18: Trayectoria del agente tras encontrarse el objetivo detrás del mismo.

Una característica interesante, la cual no fue abordada durante el desarrollo de esta tesis, es el seguimiento de objetivos móviles. Este tema es estudiado ampliamente en el ambiente de la robótica. Nuestro método fue probado en este tipo de tareas comprobándose que el mismo ha presentado un buen comportamiento en estas situaciones. En primer lugar al objetivo se le dio una velocidad constante en línea recta de forma de que el agente lo siguiera. En segundo lugar al objetivo se le dio una velocidad constante de forma de que su trayectoria sea una circunferencia, el robot fue situado en el centro de la misma.

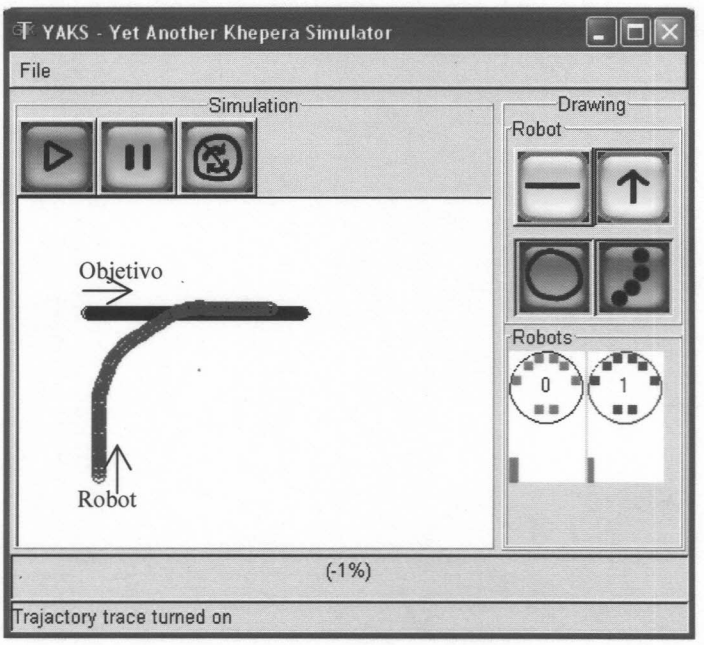


Figura 6.1.19: Trayectoria del robot siguiendo un objetivo que se mueve en línea recta

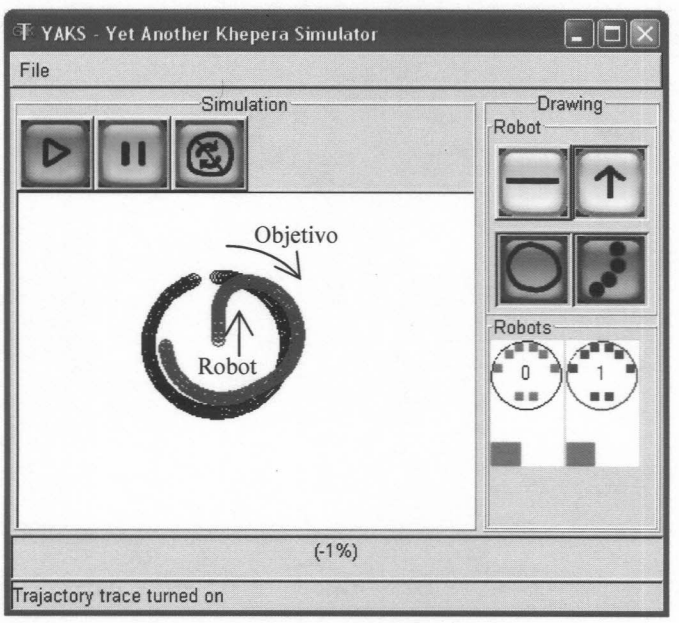


Figura 6.1.20: Trayectoria del robot siguiendo a un objetivo que se mueve en círculos.

6.2 Conclusiones

Se ha conseguido desarrollar un controlador de motor por medio de la utilización de una red neuronal del tipo mapa autoorganizado, denominada mapa autoorganizado extendido o EKM. Este modelo ha sido introducido en la robótica por diversos autores, en especial para aprender la cinemática y/o dinámica de un agente, asociando los controles de motor con los desplazamientos que pueden ser conseguidos al aplicar las velocidades correspondientes a las ruedas del robot.

Se ha desarrollado un método por medio del cual un EKM aprende la dinámica de un robot móvil. Esta información es luego utilizada mediante funciones de distancia de manera de obtener una actividad neuronal, de forma de que un objetivo genere una actividad excitatoria y los obstáculos una señal inhibitoria. Luego la actividad resultante de integrar ambas señales, excitatorias e inhibitorias, determinan la neurona del EKM que contiene el control de motor que le permite al agente dirigirse hacia el objetivo, evitando colisionar con los obstáculos detectados.

Han sido evaluadas diferentes formas de aprendizaje de la dinámica, como lo es realizar un entrenamiento anticipado a la ejecución, aprender mientras se lleva a cabo la tarea de alcanzar el objetivo o alternar dinámicamente entre ambas opciones, observándose que un entrenamiento total en forma on-line no produce un ordenamiento del mapa autoorganizado, y tampoco resulta en que los controles de motor asociados con las neuronas produzcan los desplazamientos correspondientes.

Utilizando la información contenida en el mapa autoorganizado extendido se pudo reducir la cantidad de parámetros necesarios para la detección de obstáculos y localización del objetivo. De esta forma se consiguió desarrollar una técnica que no depende de parámetros auxiliares que afectan sensiblemente el comportamiento del agente.

La ampliación de la información contenida en las neuronas para abarcar la orientación del robot, ha facilitado la obtención de un movimiento uniforme y notablemente mejor. Esta información es utilizada por el método propuesto de forma de que el robot se desplace orientándose hacia el objetivo. Por consiguiente el agente podrá sensar el entorno circundante más eficientemente, ya que posee la mayor cantidad de sensores en su frente.

Se han estudiado exhaustivamente circunstancias complejas, de forma de implementar un método eficiente y robusto que las solucione, por medio del cual se ha conseguido que un robot móvil pueda alcanzar un objetivo evitando obstáculos inclusive en circunstancias adversas.

6.3 Trabajo Futuro

En este trabajo se ha introducido la noción del tiempo en la información contenida en el mapa autoorganizado relativo a los controles de motor. De esta forma se pretende obtener información no solo del desplazamiento que es producido tras ejecutar una determinada velocidad en las ruedas del agente, sino también tener en cuenta el lapso de tiempo durante el cual estas velocidades son aplicadas. Si bien esta funcionalidad ha sido implementada, la utilización de la misma en el determinación de la neurona que posee el control de motor que será ejecutado, no ha sido tenida en cuenta, hacerlo involucra desde ya un análisis complejo.

La implementación de la técnica desarrollada en robots reales puede involucrar nuevos aspectos como ser el resbalamiento de las ruedas, fricción, y otras variables físicas que no son contempladas en simulación, el procedimiento descripto puede ser probado entonces bajo estas circunstancias, a fin de evaluar el comportamiento en ambientes reales.

Sería interesante evaluar la aplicación del método desarrollado, por ejemplo para el seguimiento de objetivos móviles o la evasión de obstáculos en movimiento. Este aporte sería de utilidad en las aplicaciones de fútbol de robots y exploración de entornos desconocidos, con el fin de seguir a un objetivo, podría incluso aplicarse en controles de automotores para el manejo automático del mismo, con el fin de seguir a otros vehículos.

Por último, se podría evaluar el comportamiento de tipos de robot distintos al Khepera, que tengan otros elementos de tracción, como por ejemplo arañas, vehículos motorizados para terrenos complejos, u otras formas de sensar el entorno circundante, sea por el tipo de sensor utilizado o por la cantidad y distribución de los mismos.

Apéndice A

Archivos de configuración del simulador YAKS

El archivo de configuración es un archivo de texto. Cada línea contiene el nombre de un parámetro y su valor correspondiente, separados por uno o más espacios. Las líneas que comienzan con el símbolo numeral (#) son consideradas comentarios. En la siguiente tabla se describen todos los parámetros aceptados.

Parámetro	Significado	Valores posibles
A_#ROBOTS	Cantidad de robots	1..n
WORLD_PATH	Directorio donde se encuentra la definición del entorno	.\worlds
WORLD_FILE	Nombre de archivo que contiene la definición del entorno	nuclear.world
SERVER_PORT	Puerto TCP/IP para utilizar en la comunicación con el controlador	1033
TEST_SAME_START_POSITION	Hace que los robot comiencen siempre en la misma posición de inicio	1: sí. 0: no.
UPDATE_DELAY	Tiempo de espera entre actualizaciones de la visualización, en milisegundos	100
VERBOSE_LEVEL	Nivel de información de debug que se imprime en la consola	0: desactivado. 3: completo
SCALE	Factor de escala de la visualización	0.5
SENSOR_NOISE	Agrega ruido a los sensores	1: sí. 0: no.
NOISE_PERCENTAGE	Cantidad máxima de ruido (porcentaje)	5
MOVE_OBSTACLES	Mueve los obstáculos aleatoriamente entre corridas	1: sí. 0: no.

El archivo de definición de entorno es un archivo de texto. Cada línea contiene las coordenadas para definir un único elemento. Las líneas inválidas son descartadas. En la siguiente tabla se describen todos los parámetros aceptados.

Nombre	Significado	Parámetros
wall	Pared	x1, y1, x2, y2
zone	Zona	x, y, r
light	Fuente de luz	x, y
roundobst	Obstáculo grande	x, y
sroundobst	Obstáculo pequeño	x, y
start	Posición de inicio	x, y, ang
Radius	Radio de todos los obstáculos	R
sradius	Radio de todos los obstáculos pequeños	R

Todo archivo de definición de ambiente debe contener al una única definición para el radio de los obstáculos (radius y sradius) y al menos tantas posiciones de inicio (start) como robots se desee utilizar (parámetro A_#ROBOTS).

Si se utiliza el parámetro MOVE_OBSTACLES (su valor es 1), la posición de los obstáculos no es tomada en cuenta.

Cada comando está formado por una letra que identifica el comando (que puede estar precedida por un modificador), una lista de parámetros que dependen de cada comando y un carácter de fin de línea. El simulador envía siempre una respuesta por cada comando recibido exitosamente. Si el comando es inválido, el simulador devuelve el mensaje Command not found.

La siguiente tabla contiene la lista de comandos aceptados por el simulador y la respuesta asociada.

Comando	Significado	Respuesta
N	Leer los sensores de proximidad	n,<s0>,<s1>,<s2>,<s3>, <s4>,<s5>,<s6>,<s7> 0 <= sn <= 1024
O	Leer los sensores de luminosidad	o,<s0>,<s1>,<s2>,<s3>, <s4>,<s5>,<s6>,<s7> 0 <= sn <= 1024
D,<left>,<right> -10 <= left, right <= 10	Asignar velocidades a las ruedas	D
E	Leer las velocidades de las ruedas	e,<left>,<right> -10 <= left,right <= 10
G,<left>,<right>	Asignas valores a los encoders de las ruedas	g
H	Leer los encoders de las ruedas	h,<left>,<right>
*<num>	Cambia el robot activo	*<num>
#E	Lee el sensor de energía	#e,<energy> 0 <= energy <= 1

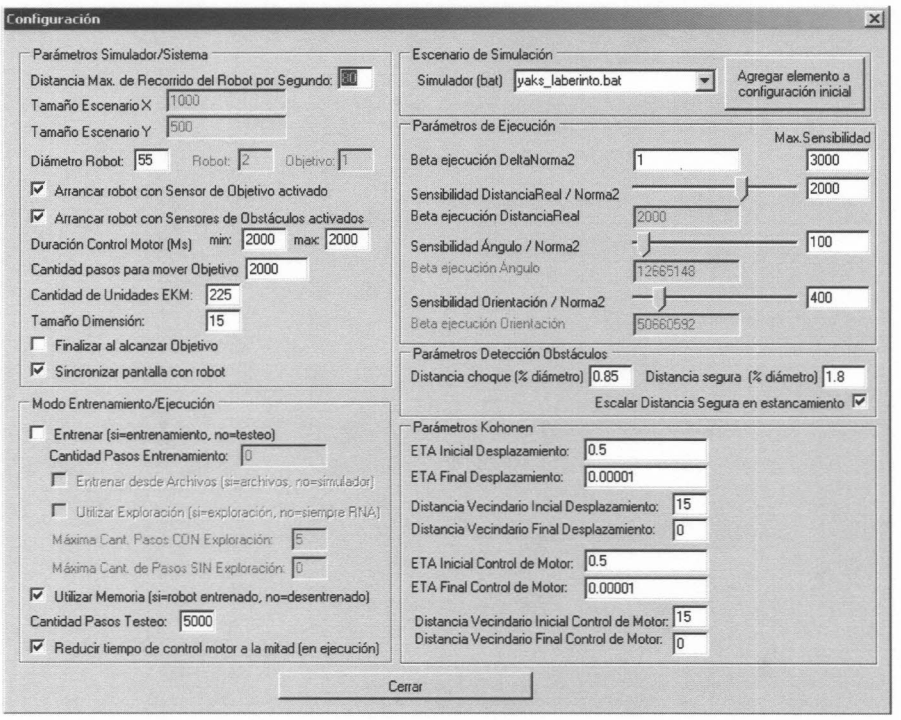
#G	Verifica si el robot se encuentra dentro de una zona	#g,<ground>
		ground = 0 o 1

#C	Lee el compas	#c,<compass> 0 <= compass <= 1

Apéndice B

Configuración del Software desarrollado

La figura siguiente corresponde a la pantalla de configuración del sistema, la misma se encuentra dividida en secciones:



- Parámetros del Simulador/Sistema: configura las variables correspondientes al simulador, como el tamaño del entorno en el cual el robot se desempeñará, el diámetro del robot, si se utilizará o no los sensores ya sea de obstáculos o de objetivo, en que lapso de tiempo se observaran los desplazamientos producidos por los controles de motor y la dimensión del Mapa Autoorganizado.

- Escenario de Simulación: especifica el nombre del archivo de configuración del simulador YAKS que será utilizado durante la simulación, puede utilizarse algunos de los existentes o agregarse uno nuevo creado por el usuario.

- Parámetros Kohonen: indica las variables propias para el entrenamiento de un mapa autoorganizado, se puede establecer la velocidad de aprendizaje así como el vecindario en el cual serán actualizadas las vecinas de la neurona ganadora.

- Modo Entrenamiento/Ejecución: establece como se utilizará el sistema, entre las opciones disponibles el usuario puede entrenar una red neuronal desde cero o utilizar una red ya entrenada, continuar con el entrenamiento de la red o sólo utilizarla para la ejecución del robot en el entorno. Si la red es entrenada sin exploración, el agente aprenderá los movimientos que ejecute mientras se mueve hacia el objetivo, si se especifican tanto los valores para exploración y sin

exploración, el robot alternará acorde a los valores establecidos entre ambas modalidades. Cuando se indica exploración se refiere a la ejecución al azar de movimientos que puede realizar el agente.

- Parámetros Ejecución: Por medio de estos controles se puede indicar y observar mientras el robot ejecuta, como el robot percibe al entorno, en base a la máxima sensibilidad configurada, los controles pueden graduarse para dar mas o menos prioridad a la detección del objetivo y obstáculos basado en la prioridad que el usuario desea, ya sea que desee dar mayor importancia al ángulo, a la orientación o la distancia.

- Parámetros Detección Obstáculos: estos parámetros modifican los dos coeficientes utilizados por el método propuesto. El escalar la distancia segura en estancamiento es una funcionalidad adicional que permite que el robot pueda salir de obstáculos demasiado complejos, que de lo contrario provocan que el agente quede oscilando sin poder evitar el mismo.

Bibliografía

[A/71] Albus J.S., *"A Theory of Cerebellar Functions"*, Mathematical Biosciences, Vol 10, 1971.

[BEF/96] Borenstein J., Everett B. and Feng L., *"Navigating Mobile Robots: Systems and Techniques"*, A K Peters, Wellesley, MA, 1996.

[CWJGL/95] Chase T.A., White M., Janet J.A., Gutierrez-Osuna R. and Luo R.C., *"Global Self-Localization for Autonomous Mobile Robots using Self-Organizing Kohonen Neural Networks"*, In Proc. of 1995 IEEE Int. Conf. on IROS, 1995.

[C/96] Collett T.S., *"Insect Navigation En Route to the Goal: Multiple Strategies for the Use of Landmarks"*, Journal of Experimental Biology, 1996.

[DGFSDC/90] Deneubourg, J. L., Goss, S., Franks, N. R., Sendova-Franks, A., Detrain, C., and Chretien, L. *"The Dynamics of Collective Sorting: Robot-like Ants and Ant-like Robots."* In Meyer, J-A, and Wilson, S., eds, Simulation of Adaptive Behaviour: from animals to animats, Cambridge, Mass.: MIT Press, 1990.

[EQCB/93] Espenschied, K.S., Quinn, R.D., Chiel, H.J. and Beer, R.D.. *"Leg coordination mechanisms in stick insect applied to hexapod robot locomotion. Adaptive Behavior"*, Adaptive Behavior, Vol. 1, 1993.

[FPB/92] Franceschini N., Pichon J.-M., Blanes C.. *"From insect vision to robot vision"*, Philosophical Transactions: Biological Sciences, Volume 337, Issue 1281, 1992.

[GR/97] Garcia-Alegre M.C. and Recio F., *"Basic Agents for Visual/Motor Coordination of a Mobile Robot"*, Proceedings of the first international conference on Autonomous Agents, 1997.

[HEK/99] Hagen S.T., l'Ecluse D. and Krose B., *"Q-Learning for Mobile Robot Control"*, In Proc. 11th BNAIC, Maastricht, 1999.

[HJRPP/05] Hernández, R., Jiménez, L.M., Reinoso, O., Puerto, R., Payá L, *"Estudio e implementación de técnicas de control reactivo de robots móviles utilizando sensores de rango (sonar e infrarrojos)"*, XXVI Jornadas de automática, 2005

[HK/00] Stepahn ten Hagen and Ben Krose, *"Q-Learning for systems with continuous state and action spaces"*, In Proc. Benelearn 2000, 10th. Bergian-Dutch Conference on Machine Learning, 2000.

[HMB/98] Huber S.A., Mallot H.A. and Bulthoff H.H., *"Modeling Biological Sensorimotor Control with Genetic Algorithms"*, Technical Report No. 060, Max-Planck-Institute for Biological Cybernetics, Tuebingen, Germany, 1998.

[HS/95] Hofner C. and Schmidt G., *"Path Planning and Guidance Techniques for an Autonomous Mobile Cleaning Robot"*, Special Issue "Research on Autonomous Mobile

Robots", Ed. Schmidt, G.. Robotics and Autonomous Systems, J. Robotics and Autonomous Systems, 1995.

[JSG/95] Jansen A., P. van der Smagt, F. Groen, *"Nested Network for Robot Control"*, Applications of Neural Networks, 1995.

[K/02] Valery Kuzmin, *"Connectionist Q-Learning in Robot Control Task"*, In Proc. proceeding of Riga Technical University, 2002.

[KE/94] Krose B.J.A. and Eecen M., *"A Self-Organizing Representation of Sensor Space for Mobile Robot Navigation"*, Proc. of the IEEE/RSJ/GI Int. Conf. on Intelligent Robots and Systems IROS'94 , Munich, Germany, 1994

[KKK/98] Kaski S., Kangas J., *"Kohonen T. Bibliography of Self-Organizing Map (SOM)"*, Neural Computing Surveys 1, 1998.

[KR/89] Kuperstein M. and Rubinstein J., *"Implementation of an Adaptive Neural Controller for Sensory-Motor Coordination"*. IEEE Control Systems Magazine, 1989.

[LKA/04] Kian Hsiang Low, Wee Khleng, Marcelo H. Ang, Jr. *"An Ensemble of Cooperative Extended Kohonen Maps for Complex Robot Motion Tasks"*. MsC Tesis, 2004.

[LKA/04] Kian Hsiang Low, Wee Khleng, Marcelo H. Ang, Jr. *"Continuos-Spacecd Action Selection fro Single and Multi-Robot Tasks Using Cooperative Extended Kohonen Maps"*. IEEE International Conference on Networking, Sensing and Control (ICNSC'04), 2004.

[LKA/03] Kian Hsiang Low, Wee Khleng, Marcelo H. Ang, Jr. *"Action Selection for Single and Multi-Robot Tasks Using Cooperative Extended Kohonen Maps"*. 18th International Joint Conference on Artificial Intelligence (IJCAI-03), 2003.

[LKA/03] K. H. Low, W. K. Leow, and M. H. Ang, Jr. *"Action selection in continuous state and action spaces by cooperation and competition of extended Kohonen maps"*. Int. Joint Conf. on Autonomous Agent & Multi Agent Systems, 2003.

[MRS/90] Martinetz T., Ritter H. and Schulten K., *"Three-dimensional neural net for learning visuomotor coordination of a robot arm"*, IEEE Trans. Neural Networks, 1990.

[POS/07] Alfredo Toriz Palacios, Maria Auxilio Osorio Lama, Abraham Sanchez Lopez, *"Planificación de rutas para robots móviles utilizando un método PRM-Genetico"*, 4° Congreso Nacional en Cs. de la Computación (Mexico), 2007

[RF/95] Rajesh P.N. Rao, Olac Fuentes, *"Perceptual Homing by an Autonomous Mobile Robot using Sparse Self-Organizing Sensory-Motor Maps"*, In Proceedings of World Congress on Neural Networks (WCNN), 1995.

[S/97] Saffioti A., *"Fuzzy Logic in Autonomous Robotics: Behavior Coordination"*, *Procs.* Of the 6th IEEE Int. Conf. On Fuzzy Systems, Barcelona, SP, 1997.

[SGG/94] P. van der Smagt, F. Groen and F. van het Groenewoud, *"Local Linear Nested Network for Robot Manipulation"*. In Proceedings of the IEEE International Conference on Neural Networks, 1994.

[SR/97] Salomon R., *"Self-Organizing Neural Network Controllers for Mobile Robots"*, Proceedings of the Artificial Neural Networks in Engineering Conference (ANNIE'97), 1997.

[TWBM/97] Trullier O., Wiener S.I., Berthoz A. and Meyer J.A., *"Biologically Based Artifical Navigation Systems: Review and Prospects"*, Progress in Neurobiology, 1997.

[VG/95] C. Versino, L.M. Gambardella. *"Learning the visuomotor coordination of a mobile robot by using the invertible Kohonen map"*. Proceedings of IWANN95, International Workhop on Neural Networks, Malaga, Spain, 1995.

[VG/00] C. Versino, L.M. Gambardella. *"Learning Fine Motion in Robotics: Design and Experiment"*, Crc Press International Series On Computational Intelligence, 2000.

[WHW/93] Walker A., Hallam J. and Willshaw D., *"Bee-havior in a Mobile Robot: The Construction of a Self-Organized Cognitive Map and its Use in Robot Navigation within a Complex, Natural Environment"*, IEEE International Conference on Neural Networks, 1993.

[WS/93] Walter J.A. and Schulten K.J., *"Implementation of Self-Organizing Neural Networks for Visuo-Motor Control of an IndustrialRobot"*. IEEE Transactions on Neural Networks, 1993.

[WR/95] Jorg Waltter, Helge Ritter, *"Investment Learning with hierarchical PSOM"*, Advances in Neural Information Processing 8. Proceedings of the 1996 Conference, 1996.

[YB/96] Yamagauchi B. and Beer R., *"Spatial Learning for Navigation in Dynamic Environments"*, IEEE Transactions on Systems, Man and Cybernetics-Part B, Special Issue on Learning Autonomous Robots, 1996.

[ZGL/95] Zalama E., Gaudiano P. and López Coronado J., A Real-time, *"Unsupervised Neural Network for the Low-level Control of a Mobile Robot in a Non-stationary Environment"*, Neural Networks, Volume 8, Number 1, 1995.