UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

POSGRADO EN CIENCIA E INGENIERÍA DE LA COMPUTACIÓN

Aplicación de Redes Neuronales y Lógica Borrosa en la Creación de Mapas Topológicos

T E S I S

QUE PARA OBTENER EL GRADO DE:

MAESTRO EN INGENIERÍA

(COMPUTACIÓN)

P R E S E N T A:

ADALBERTO HERNÁNDEZ LLARENA

DIRECTOR DE TESIS:

DR. JESÚS SAVAGE CARMONA

México, D.F. **2006.**

Aplicación de Redes Neuronales y Lógica Borrosa en la Creación de Mapas Topológicos

Adalberto Hernández Llarena

APROBADO:

Dra. Cristina Verde Rodarte	Dr. Boris Escalante Ramírez
Dr. David Rosenblueth Laguette	Dr. Angel Kuri Morales

DIRECTOR DE TESIS:

Dr. Jesús Savage Carmona

A ese pequeño ser que esta por llegar.

Agradecimientos

Al Dr. Jesús Savage Carmona por dirigir mi trabajo, por compartir su experiencia en el campo de la robótica y por su mente abierta a las nuevas ideas.

A la Doctora Cristina Verde Rodarte, a los Doctores Angel Kuri Morales, David Rosenblueth Laguette y Boris Escalante Ramírez por sus valiosos comentarios durante el desarrollo del presente trabajo y durante mi estancia en el programa de posgrado.

A las autoridades del Posgrado en Ciencia e Ingeniería de la Computación, por el excelente nivel alcanzado por el programa, tanto en lo académico como en lo administrativo.

A Lulú, Diana y Amalia, por su apoyo y acicates.

A las autoridades de la Facultad de Ingeniería, en especial al M. en I. Adolfo Millán Nájera por su apoyo y facilidades para desarrollar este proyecto.

A Sergio Cuellar Valdés por su apoyo durante las pruebas y por desarrollar el módulo principal de control del robot.

A mi esposa Betty por su poyo incondicional.

A mis padres por fomentar mi sed de conocimiento.

A mi mismo, por aceptar este reto y

A Dios por permitirme concluirlo.

Adalberto Hernández Llarena
México, D.F., México
Abril 2006

Índice

5 Creación de Mapas con Redes Neuronales

6 Aplicación de Técnicas Neuronales

Lista de Tablas

Lista de Figuras

Capítulo 1

Introducción

1.1 Motivación

Uno de los problemas principales cuando se trabaja con robots móviles lo constituye la navegación autónoma del robot. Lo anterior es necesario para que un robot móvil sea capaz de crear una representación del ambiente que lo rodea a partir de las observaciones realizadas por él mismo. Dicha representación deberá ser almacenada de alguna forma y utilizada posteriormente para localizarse y poder planear rutas que lo lleven de un punto a otro dentro del espacio de trabajo, evadiendo posibles obstáculos inesperados e inclusive realizando adaptaciones al mapa memorizado.

Esto ya se ha caracterizado con anterioridad y se conoce como el problema de *Localización y Mapeo Simultáneo* (SLAM por sus siglas en inglés) [Thrun 03], el cual pretende que un robot móvil pueda explorar el entorno de manera autónoma con sus sensores, obtener conocimiento acerca de éste, interpretar la escena, construir un mapa apropiado y localizarse a sí mismo en relación a dicho mapa de manera simultánea.

1.2 Descripción de la Problemática

El problema anterior implica dos cuestiones: la primera la constituye la generación de un “mapa topológico”, es decir, una serie de nodos que representen distintas zonas o ubicaciones posibles de un robot dentro de un espacio dado, y la segunda el problema de determinar en qué posición de dicho mapa el robot se encuentra en realidad.

El presente trabajo se aboca a la tarea de crear la representación del entorno del robot en forma de algún tipo de mapa y de la forma de almacenamiento de la misma. Sin embargo, se aborda brevemente el problema de la localización del robot.

Actualmente es común el uso de sonares como sensores. No obstante se sabe que dichos dispositivos presentan demasiados problemas debido a la absorción y reflexión de las señales emitidas por los mismos. Lo anterior hace difícil su uso para la tarea de creación de mapas y ha llevado al uso generalizado de sensores de láser por su mayor confiabilidad. Sin embargo, el alto costo de este último tipo de dispositivos hace imprescindible contar con métodos que garanticen la confiabilidad de las mediciones realizadas por los sonares

para aplicaciones medianas y pequeñas cuyo costo deba ser tomado en cuenta de una manera importante.

Por otro lado existe desde luego un problema previo a la obtención de lecturas: ¿cómo hacer que el robot de manera autónoma explore el ambiente, sin colisionar contra objetos y recorra la mayor parte del espacio navegable?, si bien es posible desarrollar un algoritmo que, con base en las lecturas proporcionadas por el o los sonares, indique los movimientos sucesivos del robot, en la práctica, dados los problemas con las lecturas de los sonares así como la cantidad y variedad posible de ambientes de navegación, el desarrollo de dichos algoritmos o funciones de navegación se vuelve extremadamente complejo y requiere mucho tiempo para ser evaluado y modificado.

De esta forma se vuelve imprescindible el uso de técnicas que tomen en cuenta las limitantes en el funcionamiento de los sonares, eliminen las lecturas erróneas y proporcionen datos fidedignos tanto para ser utilizados tanto en la navegación autónoma como en la creación del mapa del entorno.

En este caso se pretende explorar el uso de las herramientas conocidas como *redes neuronales artificiales* por sus capacidades de aprendizaje y *lógica borrosa* por tratar con grados de certeza y confiabilidad.

1.3 Objetivos de la Tesis

a) Desarrollar un método basado en redes neuronales que permita a un robot móvil:

1) **Navegar libremente por el ambiente de trabajo.**
2) **Elaborar una representación de su entorno, con base en las lecturas proporcionadas por sus sonares durante el recorrido, tomando en cuenta los problemas que se pueden presentar con tales mediciones.**

b) Evaluar el desempeño de dicho método con un robot y ambiente real.

1.4 Contribución y Relevancia

Es sabido que uno de los principales problemas con el uso de redes neuronales se presenta cuando existen errores en las mediciones asociadas (vector de entrada) de la red y que, una vez entrenada la misma (que dicho sea de paso constituye en sí mismo un gran problema en cuanto a la determinación del conjunto de entrenamiento, topología y número de épocas de aprendizaje), el proceso de re-adaptación resulta difícil y en ocasiones es preferible diseñar una nueva red que se adapte a los nuevos requerimientos.

Por tal motivo se busca analizar y diseñar una estrategia que permita a la red no ser tan susceptible a variaciones en los datos de entrada (como las debidas a la presencia de obstáculos pequeños o errores en los sonares), por medio del preprocesamiento de los datos alimentados a la red y mediante la adición de características de lógica borrosa que le permitan lidiar con grados de certeza.

De encontrarse un esquema apropiado, éste podría constituir una contribución para los sistemas de navegación y posicionamiento robótico ya que tornaría mucho más factible y confiable el uso exclusivo de sonares en la creación de mapas del entorno. Por otro lado se pretende probar las capacidades prácticas de aprendizaje de las redes neuronales para problemas de navegación autónoma.

1.5 Vista General de la Tesis

El capítulo 2 introduce el concepto de navegación en robots móviles: los diferentes tipos de robots móviles, los problemas asociados a la navegación y los diferentes tipos de mapas y medios para crearlos.

El capítulo 3 proporciona las bases necesarias para entender las redes neuronales artificiales y expone los principales tipos de redes y sus aplicaciones en particular.

El capítulo 4 hace referencia a los principales conceptos de la Lógica Borrosa y sus aplicaciones prácticas.

El capítulo 5 se adentra directamente en el problema de la creación automática del mapa y del caso de estudio en particular, donde se pretende que un robot móvil genere de manera autónoma dicho mapa.

El capítulo 6 muestra un caso práctico donde son puestos a prueba los métodos utilizados, se evalúa el desempeño de los mismos en un ambiente controlado y reporta los resultados de los métodos desarrollados.

Finalmente dentro del capítulo 7 se hace un resumen de la tesis y se presentan las áreas de oportunidad de mejora para trabajos futuros.

Capítulo 2

Navegación en Robots Móviles

2.1 Robots Móviles

"Un robot inteligente es una máquina capaz de extraer información de su medio ambiente y utilizar dicho conocimiento sobre su mundo para moverse con seguridad, de forma significativa y con un propósito" [Arkin 98]

"... La revista Newsweek lo resumía como: «Hoy en día ya se puede comprar algo que se parece a R2D2. Pero este R2D2 no nos ayudará a rescatar a la princesa Leia. De hecho no nos serviría ni para sacar la basura»" [Pawson 86]

Existe una gran cantidad de definiciones para el término *robot* y muchas más para el tipo de acciones que debe realizar, pero la mayoría coincide en un aspecto: *movilidad*.

Resulta difícil de imaginar un robot que no posea ninguna capacidad de movimiento ya que indudablemente la movilidad siempre ha formado parte de las expectativas que se tienen y se desean de los robots.

Pero, ¿cuáles son las acciones que se esperan de los robots móviles?, principalmente *desplazarse* dentro de un ambiente real que puede ser algún terreno, el agua o inclusive el aire. De esto se intuye que será necesario que el robot cuente con un sistema de locomoción que le haga posible transportarse; puede poseer ruedas, orugas o cualquier sistema que le permita cambiar su posición física en el medio ambiente. Sin embargo esto no será suficiente, pues el robot debe poseer una idea sobre su nueva *posición* y para percibir dicha nueva posición deberá contar con sensores que le brinden información sobre la misma.

Asociado al concepto de *posición* está la cuestión *con respecto a qué*, y es en este punto en donde entran los conceptos de *mapa* y *localización*. Existe una problemática ampliamente conocida como *localización y mapeo simultáneo* (SLAM por sus siglas en inglés) [Thrun 03] que formula el problema que implica para los robots móviles navegar de forma autónoma por un ambiente desconocido, e ir creando de manera simultánea el mapa del sitio, mediante las lecturas de sus sensores y mantenerse al tanto de su nueva posición dentro del mapa creado en cada momento.

Existen diversos tipos de robots móviles. Hoy en día los adelantos en las áreas de percepción, control y tecnología de movimiento han permitido que se implementen sistemas robóticos "inteligentes" en áreas distintas a la producción industrial. A mediano y largo plazo los expertos en robótica prevén que el número de robots de servicio (aquellos utilizados para las tareas que comúnmente son realizadas por humanos) superará al número de robots industriales. El desarrollo se está acelerando en esta área. La innovación, y en particular la creatividad, son las fuerzas que guían este proceso. Por tal motivo será imprescindible contar con métodos que garanticen contar con robots de una alta eficiencia y muy bajo costo [Schraft 00].

Entre las nuevas áreas de aplicación de los robots móviles se incluyen: agricultura, construcción, limpieza, oficina, combate al fuego, recreación, medicina e inmersión, entre otras.

Figura 2.1: Distintos tipos de robots móviles.

2.2 ViRbot

Los robots móviles deben tener dos capacidades básicas: *adaptabilidad* (para reaccionar en forma oportuna y apropiada a sucesos imprevistos y modificaciones de su medio) y *determinación* (para escoger las acciones apropiadas que les permitan lograr sus objetivos). Además deben ser capaces de solucionar problemas relacionados con la adaptación al medio o con la determinación para tratar de lograr metas de manera eficiente y aplicar procedimientos probados en situaciones rutinarias.

En [Savage 98] se aborda el tema de los diferentes módulos que deben integrar un robot móvil:

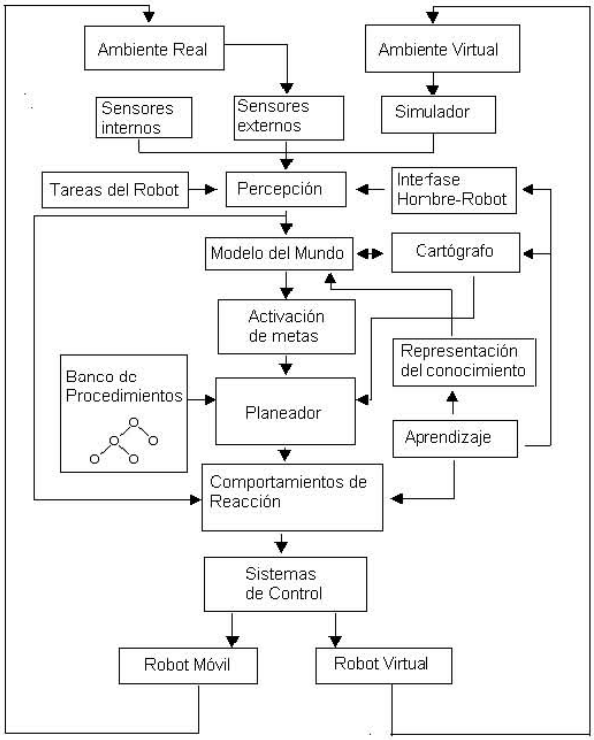

Figura 2.2: Esquema ViRbot de los módulos de un robot móvil.

2.2.1 Ambiente real.- Mejor conocido como el mundo real, el ambiente real consiste en la parte física (paredes, objetos, luces, sonidos, etc.) con la cual un robot interactúa y que no se encuentra restringido e idealizado, por ejemplo, el ambiente real de operación de un robot puede ser una fábrica, un edificio o un taller. El ambiente real es el lugar de operación del robot donde está expuesto a todo tipo de situaciones imprevistas.

2.2.2.- Ambiente virtual.- El ambiente virtual consiste en una idealización del ambiente real de operación del robot, a menudo consiste en un ambiente simulado o ambiente controlado donde se pueden agregar uno a uno los factores externos con los que interactúa el robot (fricción, errores de movimiento u operación, nuevos elementos u objetos, iluminación, temperatura, etc.). Las acciones de los robots son perfeccionadas primero actuando en ambientes virtuales, para luego ser probadas en el mundo real.

2.2.3.- Simulador.- Cuando se trabaja con ambientes virtuales es necesario poseer un modelo que simule la realidad, es decir, que permita observar qué sucede cuando el robot ejecuta cierto tipo de acción y desde donde pueda obtener información como si se tratase del mundo real, en la medida que la simulación se asemeje a la realidad, más sencillo será implementar procedimientos dentro del robot que funcionen bien bajo condiciones reales de operación. Por ejemplo, si se modelan las ecuaciones de propagación de ondas para los sensores o las ecuaciones de movimientos de los motores, la simulación será más realista.

2.2.4.- Sensores.- Los sensores constituyen la interfaz de entrada al robot, es decir, le permiten obtener información del mundo externo (ya sea real o virtual). Entre los tipos de sensores se citan los siguientes:

a) Sensores de distancia (ultrasonido, infrarojo, láser, etc.).
b) Sensores de visión (cámaras).
c) Sensores de contacto.
d) Sensores auditivos (micrófonos).

2.2.5.- Condiciones internas del robot.- Algunos tipos de sensores están enfocados al interior del robot, es decir, permiten hacer mediciones de información sobre la operación interna del robot, ejemplos de ello son los medidores de nivel de batería, contadores de vueltas de las llantas, giroscopios o brújulas.

2.2.6.- Interfaz hombre-robot.- Una interfaz hombre-robot permite una comunicación entre el robot y un ser humano, por ejemplo un módulo de procesamiento del lenguaje natural le permitiría al robot recibir comandos hablados por parte de un operador.

2.2.7.- Tareas del robot.- Las tareas que el robot debe realizar son especificadas mediante un guión o *script*. Es posible utilizar un módulo que haga uso de dependencia conceptual [Schank 72] para implementar los procedimientos a realizar. Dependencia conceptual permite representar una tarea específica, por ejemplo “robot ve a la cocina”, mediante una o más primitivas de acción con la forma:

(<nombre primitiva> (actor *actor*)(object *objeto*)(from *origen*)(to *destino*))

Un ejemplo de ello es la primitiva *Ptrans* que transfiere físicamente un objeto de posición (sinónimo de *trasladar*). Para el ejemplo citado la representación sería

(Ptrans (actor robot)(from robot)(to kitchen))

Cabe aclarar que en este caso (from robot) indica la posición actual del robot.

2.2.8.- Sistema de percepción.- A partir de la información (en bruto) proporcionada por los sensores (por ejemplo la información relativa a imágenes de entrada o sensores de distancia), la interfase hombre-robot (lenguaje natural) y las tareas del robot (dependencia conceptual), el módulo de percepción obtiene representaciones simbólicas de ellos y genera una creencia.

2.2.9.- Modelo del mundo.- En esta etapa se validan las creencias generadas por el sistema de percepción e interactuando con el cartógrafo.

2.2.10.- Cartógrafo.- Posee una representación del mundo, a menudo consiste en un mapa que puede ser geométrico (polígonos), simbólico (pared, lisa) o topológico, cuando el modelo del mundo valida una creencia (a menudo relacionada con las condiciones externas del mundo) el cartógrafo actualiza su información y agrega o elimina los elementos representados.

2.2.11.- Activación de metas globales y locales.- Una vez que se han validado las creencias se activan objetivos específicos (por ejemplo movimientos) encaminados a realizar las metas señaladas.

2.2.12.- Soluciones alambradas.- Consisten en secciones de código que resuelven problemas específicos (por ejemplo tomar un objeto, girar cierto ángulo o avanzar cierta distancia, etc.), por lo general consisten en máquinas de estados que realizan cierta función, sin embargo, si el robot cuenta con un módulo de aprendizaje, una solución alambrada puede consistir en una secuencia de acciones que el robot ha aprendido previamente.

2.2.13.- Planeador.- Con base en las metas específicas, este módulo decide qué procedimientos con los que cuenta de antemano va a utilizar para estructurar una secuencia de acciones que permita resolver los objetivos globales o locales, como ejemplo, si se trata de acciones de desplazamientos del robot, encuentra el camino a seguir para llegar a dicha posición.

Este módulo busca secciones de código que resuelvan problemas parcialmente.

2.2.14.- Navegador.- Una vez que se tiene la ruta que el robot debe seguir (puntos a visitar) el navegador encuentra las ecuaciones de movimiento: velocidad, aceleración, distancia y giro que deben aplicarse para trasladar al robot a cada una de las posiciones establecidas.

2.2.15.- Representaciones del conocimiento.- Consiste en una codificación del conocimiento mediante reglas y hechos (por ejemplo dentro del sistema experto CLIPS o prolog). Está en contacto permanente con el modelo del mundo ya que hechos o reglas específicos pueden estar relacionados con las condiciones del mundo real.

2.2.16.- Aprendizaje.- Este módulo permite que procedimientos elaborados exitosamente y situaciones previas pasen a formar parte de la base de conocimiento (por ejemplo un robot que no tenga mapa y pueda generar dicho mapa con observaciones propias).

2.2.17.- Piloto.- Ejecuta los comandos dados por el navegador, hace que gire el robot y avance, controla los motores mediante máquinas de estados y recibe permanentemente información de los sensores.

2.2.18.- Métodos de comportamiento.- Consisten en procedimientos que son activados cuando se generan ciertas situaciones, por ejemplo obstáculos inesperados, en este caso es necesario ejecutar un comportamiento que permita evitar el obstáculo, por ejemplo el algoritmo BUGS que sigue los contornos del obstáculo para evitarlo.

2.2.19.- Algoritmos de control.- Permiten controlar dispositivos de hardware con exactitud mediante máquinas de estados, filtros adaptables, control analógico y digital retroalimentado o en lazo abierto, etc. Los algoritmos de control pueden ser aplicados tanto al robot real como al robot virtual cuando se está simulando el ambiente real.

2.2.20.- Robot virtual.- Simula el agente sobre el cual se aplican los algoritmos de control y movimiento ejecutados por el piloto.

2.2.21.- Robot Móvil.- Robot real que se encuentra en contacto con el mundo externo.

Una vez que se han ejecutado acciones sobre el robot real o virtual, el resultado de sus acciones modifica el entorno real o virtual, dando origen a nuevas percepciones que a su vez generan nuevos comportamientos y así sucesivamente.

2.3 Navegación

"*Una de las principales metas en robótica es crear robots autónomos. Tales robots deberán aceptar descripciones de alto nivel de tareas y deberán ejecutarlas sin ayuda humana extra. Las descripciones de entrada especifican qué es lo que el usuario quiere pero no cómo hacerlo*" [Latombe 91].

Desarrollar las tecnologías necesarias para los robots autónomos resulta una tarea formidable, con múltiples ramificaciones intrincadas entre razonamiento automático, percepción y control. Involucra muchos problemas importantes. Uno de ellos es navegación que puede plantearse de la siguiente forma: ¿cómo puede un robot decidir qué movimientos debe ejecutar para ordenar una serie de objetos físicos?, esta capacidad es necesaria ya que, por definición, un robot completa tareas moviéndose en el mundo real. Lo mínimo que se espera de un robot real autónomo es la capacidad de planear sus movimientos.

Una idea común aunque errónea es que la navegación y planeación de movimientos consiste únicamente en hacer un poco de verificación de colisiones y evitar obstáculos. En la práctica es mucho más que eso. Involucra aspectos como planeación de rutas libres de obstáculos entre objetos posiblemente en movimiento, coordinar movimientos de varios robots, razonamiento sobre la incertidumbre para crear estrategias confiables de movimiento basado en sensores, lidiar con modelos físicos de masa, gravedad y resistencia. Por lo tanto la navegación requiere que el robot considere tanto restricciones geométricas como restricciones físicas y temporales. En adición, la incertidumbre le debe requerir planear no sólo comandos de movimiento, sino su interacción con los sensores. Cuando exista poco conocimiento al momento de planear las rutas, se tornará necesario intercalar la planeación y la ejecución con el fin de recabar la información apropiada mediante los sensores."

Con lo anterior se espera que quede claro que para realizar las tareas de navegación se debe trabajar conjuntamente con planeación, percepción y localización con cierto grado de incertidumbre.

2.4 Planeación

Existen gran cantidad de métodos para resolver el problema de la planeación de movimientos, aunque no todos resuelven el problema en su forma más general. Por ejemplo, algunos métodos requieren que el espacio de trabajo sea de dos dimensiones y los objetos polígonos. En su forma general, estos métodos se basan en tres tipos de aproximaciones: *mapa de rutas, descomposición en celdas y campos potenciales*.

2.4.1 Mapa de Rutas

Un mapa de rutas consiste en almacenar la conectividad del espacio navegable del robot en una red de curvas unidimensionales llamadas el mapa de rutas (*roadmap*) o red de conectividad. Una vez que ha sido construido dicho mapa, la planeación se reduce a conectar los puntos de inicio y final de movimiento mediante una ruta de dicho mapa.

Se han propuesto diversos métodos basados en esta idea general. Dichos métodos calculan diferentes tipos de mapas llamados grafo de visibilidad, entre los que destacan los diagramas *Voronoi*, red de carreteras (*freeway*) y de silueta.

En [Latombe 91] puede verse una descripción más completa de cada método.

2.4.2 Descomposición en Celdas

Los métodos de descomposición en celdas son quizás los métodos de planeación más ampliamente utilizados y estudiados. Consisten en descomponer el espacio libre del robot en regiones simples llamadas *celdas*. A continuación se construye un grafo no-dirigido representando la relación de adyacencia entre las celdas, llamado grafo de conectividad. Sus nodos son celdas extraídas del espacio libre y dos nodos están conectados por una liga si, y solo si, las dos celdas correspondientes son adyacentes. El resultado de la búsqueda es una secuencia de celdas llamada un *canal* (channel). Una ruta libre continua pueda calcularse a partir de esta secuencia.

Los métodos de descomposición en celdas pueden dividirse en métodos *exactos* y *aproximados*. Los métodos exactos descomponen el espacio libre en celdas cuya unión corresponde exactamente con dicho espacio libre, mientras que la descomposición aproximada produce celdas de forma predefinida (por ejemplo rectángulos) cuya unión está estrictamente incluida dentro del espacio libre.

2.4.3 Campos Potenciales

Una aproximación a la planeación de movimientos consiste en discretizar el espacio de trabajo en una fina malla regular y buscar en esta malla una ruta libre. Esta aproximación requiere de heurísticas poderosas para guiar la búsqueda ya que la malla por lo general es enorme. Varios tipos de heurísticas se han propuesto. Las más exitosas asumen la forma de funciones que se interpretan como *campos potenciales*. Ésta es una de las técnicas más utilizadas para controlar robots móviles [Latombe 91, cap. 7]. En estos casos, el entorno esta representado por un campo de potencial bidimensional (si existen muchos grados de libertad a controlar, será necesario utilizar campos de potencial de más de dos dimensiones). El campo de potencial se obtiene sumando las componentes “atractivas” y “repulsivas”. La componente atractiva representa la posición objetivo. La función

$$p_a(X) = k_1 d(X)^2 \qquad (2.1)$$

es una típica función atractiva, donde $d(X)$ es la distancia euclidiana entre el punto X y el objetivo mientras que k_1 representa una constante de atracción. Dicha función toma el valor mínimo 0 en la posición objetivo. Los componentes repulsivos representan obstáculos en el entorno. La función

$$p_r(X) = k_2/d_0(X)^2 \qquad (2.2)$$

es una típica función repulsiva, donde $d_0(X)$ representa la distancia euclideana entre el punto X y el punto más cercano del obstáculo.

El potencial total se calcula mediante la suma

$$p(X) = p_a(X) + p_r(X) \qquad (2.3)$$

y el movimiento del robot se realiza a través de la dirección señalada por el gradiente del campo de potencial o, dicho de otra forma, deslizándose por las pendientes de dicho campo. El campo potencial puede ser precalculado y almacenado en la memoria del robot a modo de modelo del mundo o se pude calcular incrementalmente respecto a la posición del robot cuando éste necesite determinar el siguiente movimiento. Sin embargo, el método puede verse afectado por la existencia de mínimos locales de potencial que podrían atrapar al robot. En [Latombe 91] se pueden analizar algunos métodos para solucionar este problema.

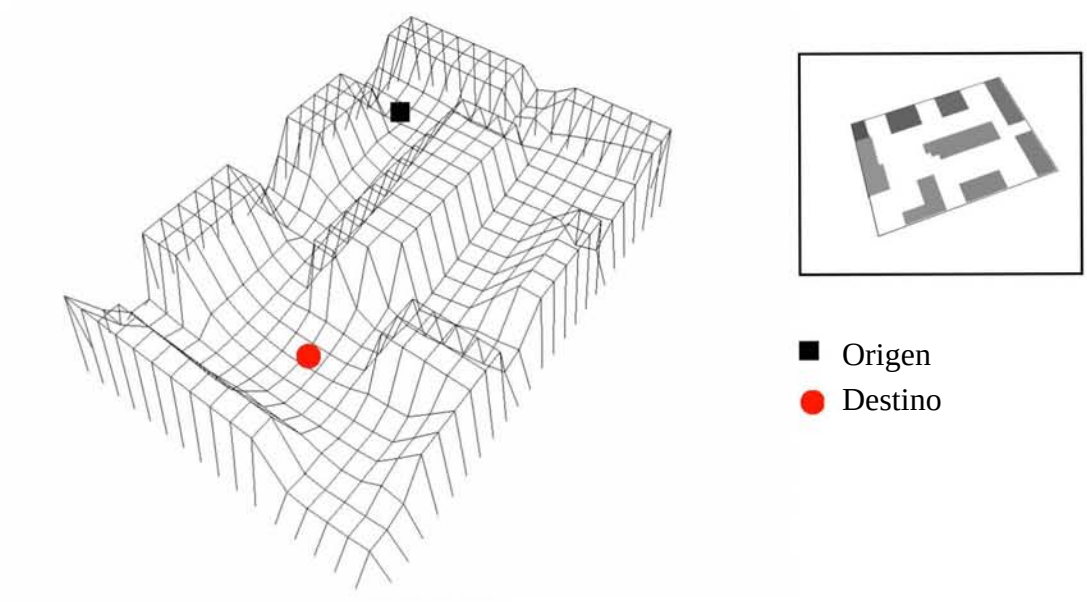

Figura 2.3: Mapa de campos potenciales (izquierda).

Cada objeto (derecha arriba) representa una elevación y el destino una depresión.

2.5 Percepción

Según [Dudek 00] la percepción es un requerimiento clave inclusive para el comportamiento más simple de un robot móvil. El robot debe ser capaz de obtener información del medio y de tomar decisiones con base en la información de sus sensores. Los sensores fundamentales asociados a la movilidad son aquellos que miden la distancia que las llantas han recorrido por el piso, sensores que miden los cambios internos y aquellos que miden la estructura externa en el medio ambiente. Para que un robot identifique dónde está o cómo ha llegado ahí se requiere de sensores y de algoritmos de percepción.

Los sensores y los algoritmos para interpretar tales parámetros pueden ser en extremo complejos. No obstante, existen una gran cantidad de sensores, es posible separarlos en dos clases: *sensores visuales*, que utilizan luz reflejada procedente de los objetos en el medio ambiente y *sensores no-visuales*, que utilizan audio, inercia y otras modalidades para percibir el medio ambiente. En el presente trabajo se describen los sensores no-visuales exclusivamente.

2.5.1 Conceptos Básicos

Quizás la clasificación más fundamental para los tipos de sensores no-visuales es sensores de *estado interno* y sensores de *estado externo*. Los sensores de estado interno proveen realimentación sobre los parámetros internos del robot como el nivel de la batería, posición de las ruedas o los ángulos en las juntas de una pierna o brazo del robot. Los sensores de estado externo tratan con la observación de aspectos exteriores al propio robot: humedad, color de los objetos, etc. De entre ellos se conoce a aquellos que trabajan cuando están en contacto con algún objeto como *sensores de contacto*, mientras que el resto se considera de *no-contacto*.

También se dice que un sensor es *activo* o de emisión, si transmite algún tipo de señal al medio, que éste altera y luego recibe nuevamente el sensor para interpretarla, o si opera modificando directamente el ambiente. Un sensor es *pasivo* o receptor si únicamente toma las señales provenientes del propio medio ambiente (lumínicas, acústicas, etc.) para realizar sus observaciones.

Es posible clasificar a las diferentes tecnologías de sensores de acuerdo a muy diversas propiedades, entre las que destacan: velocidad de operación, costo, rango de error, robustez, requerimientos computacionales, de energía, peso y tamaño.

2.5.2 El Sonar

El sonar (*sound navigation and ranging*) utiliza señales acústicas para efectuar sus lecturas. El sonar es un sensor de emisión ya que una señal o pulso es emitido y subsecuentemente la reflexión es recibida. El tiempo entre estos eventos, el cambio en la fase de la señal recibida y la atenuación de la misma en función de la frecuencia son aspectos de la señal reflejada que han sido explotadas por diferentes tipos de sensores de este tipo.

Varias unidades de sonar típicamente se instalan en robots móviles ubicadas en posiciones específicas alrededor de la base. Una estrategia utilizada consiste en disponer de un anillo de sonares situados a intervalos regulares dentro de la circunferencia del anillo. Otra estrategia implica colocar un emisor y receptor único sobre una plataforma giratoria en la parte superior del robot de tal suerte que se puedan realizar mediciones en cualquier dirección.

Los sensores de sonar utilizados en robots móviles tienden a utilizar estrategias simples de cálculo. La estrategia más simple consiste en utilizar el mismo transductor como emisor y receptor emitiendo un pulso corto de sonido para medir el tiempo de retardo hasta que el eco es recibido. Debido a oscilaciones residuales en el transductor inmediatamente después de emitir la señal, no es posible detectar con confiabilidad ecos entrantes en un intervalo breve después de que la señal de salida ha sido generada; esto se conoce como el problema de "intervalo de ceguera" (*blanking interval*). Como resultado, los transductores del sonar no detectan obstáculos que se encuentran muy cerca de la superficie del transductor, típicamente alrededor de 6 cm. Sin embargo, el valor real dependerá de cada implementación.

Si las ondas del sonar viajan directamente a un objeto, lo alcanzan y regresan directamente hacia el transductor, la distancia del transductor al objeto es la mitad de la distancia total recorrida por la onda sonora según la expresión

$$d = 1/2\ ct \tag{2.4}$$

donde d es la distancia al objeto, c es la velocidad del sonido en el aire y t es el intervalo de tiempo entre la transmisión y recepción de la señal.

La velocidad del sonido en el aire puede ser aproximada por

$$c = c_0 + 0.6T \tag{2.5}$$

donde T es la temperatura en grados celsius y c_0 es 331 m/s.

"Una vez que la onda ha dejado el transductor, el frente de onda se asemeja una porción de una esfera en expansión y por lo tanto la amplitud de la señal decrece con el cuadrado de la distancia. Esta dispersión generalmente prevalece pasado un metro de distancia de la superficie del transductor. La distribución de energía en esta superficie esférica no es uniforme, con una amplitud que varía dentro de una envolvente en función del ángulo con respecto a la normal a la superficie del transductor. Mientras más viaje la señal, más atenuada estará y los ecos tardarán más en llegar y serán más tenues. Típicamente esto es compensado en parte aumentando la ganancia del receptor de sonar en función del tiempo de espera del eco de la señal emitida. Un factor adicional que afecta la amplitud del eco recibido es la reflectancia acústica del objeto. Las reflectancias típicas varían de 98% para el concreto a solamente 30% para un plafón acústico" .

[Dudek 00, pp. 57]

"Las frecuencias utilizadas en aplicaciones comerciales de sonar casi invariablemente caen en el rango de 40 a 50 Khz. Las frecuencias más altas son atenuadas más rápidamente aunque proporcionan mejor resolución espacial. La señal emitida por el sonar en sí misma posee la forma de un cono en 3 dimensiones con distribución no-uniforme de energía sobre el cono. La energía máxima se encuentra en el centro con múltiples lóbulos laterales. En consecuencia el diámetro efectivo del cono decrece en función de la distancia. Una implicación de este frente de onda es que el eco recibido puede ser la reflexión de una porción de la onda que deja el transductor en ángulo oblicuo. De esta forma un eco único registrado no resulta necesariamente en una precisa localización del objeto utilizando únicamente métodos de cálculo de tiempos de retardo.

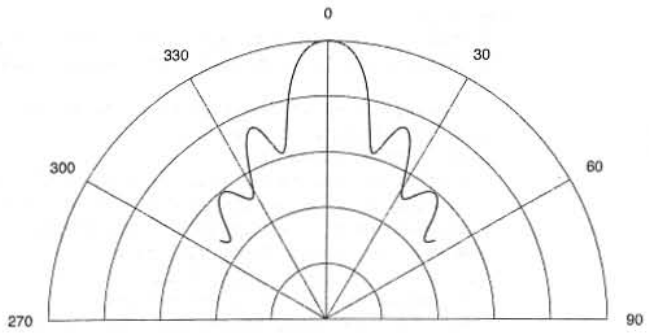

Figura 2.4: Amplitud de la señal como función del ángulo de desviación de la normal a la superficie del transductor para un emisor de cámara *Polaroid* (tomado de [Dudek 00]).

El impedimento más serio para inferir la ubicación de una superficie reflejante de las ondas del sonar es que, a frecuencias ultrasónicas, los objetos más comunes se comportan como *reflectores especulares*; esto es, exhiben propiedades similares a los espejos (para el caso de señales lumínicas) donde una señal acústica oblicua será reflejada alejándose del receptor en lugar de dirigirse hacia él. Si un eco regresa al receptor es posible asegurar que el eco no es el resultado de una compleja serie de reflexiones a lo largo del ambiente, en lugar de ser la reflexión del objeto más cercano en la dirección que apunta el transductor. En adición, los objetos con una sección transversal mucho menor que la longitud de onda utilizada regresaran muy poca energía y parecerán casi invisibles".

[Dudek 00, pp. 58]

Para una descripción más completa sobre el sonar refiérase a [Curtis 87].

2.6 Localización

Para poder realizar numerosas tareas un robot móvil necesita saber "dónde está". Un robot puede necesitar saber su posición exacta para poder planear rutas apropiadas o para saber si su posición actual es la adecuada para realizar ciertas tareas. Lo anterior implica estimar la posición del robot (en términos cualitativos o cuantitativos) con respecto a alguna representación global del espacio: es posible referirse a esto como localización fuerte (*strong localization*). El problema de la localización débil (*weak localization*) en contraste, consiste únicamente en saber si la ubicación ha sido visitada con anterioridad. En ciertos casos mapas cualitativos completos pueden construirse a partir de localización débil.

Dado un estimado aproximado de la posición del robot basado en odometría, un mapa del medio ambiente y suficientes datos de los sensores, es posible mantener un constante estimado de la ubicación del robot con respecto al mapa. Este proceso es conocido algunas veces como *localización*, *estimación de la posición* o *posicionamiento*. La especificación general de este problema comienza con un estimado inicial de la ubicación del robot $\boldsymbol{q}$ dada por una distribución de probabilidad $P(\boldsymbol{q})$. La localización basada en sensores se funda en la premisa de son utilizados los datos del sensor s en conjunción con un mapa para producir un estimado de la posición $P(\boldsymbol{q}|s)$ tal que dicho estimado tenga una mayor densidad de probabilidad sobre la posición verdadera del robot.

En ciertas circunstancias puede ser necesario inferir la posición del robot sin un estimado a priori de su ubicación. Este tipo de posicionamiento es referido como localización global. Una versión más común del problema de localización consiste en la necesidad de refinar un estimado de la posición continuamente. Esta tarea se conoce como *mantenimiento de la posición* o *localización local*.

Quizás la aproximación más simple al problema del mantenimiento de la posición consiste en utilizar control de lazo abierto (manteniendo un modelo del mundo basado en los resultados esperados de las acciones sin realizar observaciones del mundo exterior) u odometría para llevar registro de qué tanto se ha movido el robot en cada dirección y entonces sumar los desplazamientos para producir un desplazamiento en la red que puede ser agregado al estimado inicial de la posición. Calcular la distancia que el robot se ha movido sin ninguna referencia al mundo exterior se conoce como "estimación muerta" (*dead reckoning*). Esta es la técnica utilizada para medir la distancia recorrida por un automóvil utilizando un odómetro.

Esencialmente todas las técnicas para estimar la posición deben de lidiar con errores debidos a ruido eléctrico, cuantificación, digitalización, deslizamiento de las ruedas, movimientos en los engranes, etc. Si se utiliza exclusivamente *dead reckoning* para estimar la posición, estos errores se sumarán al estimado de la posición y se irán acumulando con movimientos sucesivos del robot, lo cual convierte al proceso de localización en una tarea imposible si no se cuenta con información externa que permita eliminar sucesivamente los errores. Para los problemas de localización de largo plazo asociada a tareas de navegación y construcción de mapas debe tomar referencias del mundo exterior para corregir y mantener su posición actualizada.

Un paso clave en el proceso de la localización tanto global como local consiste en comparar el conjunto de observaciones actuales contra un mapa establecido. Los métodos de comparación pueden ser clasificados en las siguientes categorías:

- **Comparación Datos-Datos**. En la comparación datos-datos se comparan directamente los datos en crudo con los datos crudos predichos (extraídos del mapa ya sea por modelación predictiva o utilizando conjuntos de datos almacenados).

- **Comparación Datos-Modelo**. La comparación datos-modelo compara los datos observados con modelos más abstractos almacenados en el mapa (basados en un modelo de cómo se asocian los modelos con los datos).

- **Comparación Modelo-Modelo**. La comparación modelo-modelo compara los modelos almacenados en el mapa contra los modelos generados a partir de las observaciones actuales.

Cada una de las técnicas ha sido utilizada con cierto grado de éxito, y cada una de ellas tiene un dominio de aplicabilidad, dependiendo particularmente de las características de los sensores utilizados y del método de adquisición de datos.

2.7 Mapas

Los mapas usualmente representan elementos estructurales en algún sentido abstracto (quizás mediante etiquetas semánticas), dentro del cual el robot móvil debe ser capaz de relacionar su ubicación actual directamente con sus propias observaciones de este medio ambiente.

En realidad para un robot construir por sí mismo el mapa de su medio ambiente es una tarea increíblemente difícil y tediosa. No obstante realizar mediciones de una sola habitación puede ser realizado sin muchos problemas, reconstruir con precisión un mapa métrico de un ambiente interior grande resulta difícil de manejar debido a que puede no existir una línea clara de visión entre características prominentes del medio ambiente.

Los mapas pueden tomar muchas formas. Dos representaciones específicas resultan particularmente relevantes para los robots móviles: *mapas métricos* que están basados en un marco de referencia absoluto y estimados numéricos de dónde se encuentran los objetos en el espacio y *mapas topológicos* (también conocidos como mapas relacionales) que sólo representan explícitamente información, típicamente en forma de grafo.

Para poder explotar las ventajas de las representaciones métricas y topológicas resulta a menudo apropiado considerar la construcción de alguna representación utilizando observaciones sobre alguna representación menos abstracta. Esto conlleva a un arreglo jerárquico de representaciones sucesivas de los datos utilizados para los mapas como el que se muestra a continuación según [Dudek 00]:

1. **Sensorial**. Señales de datos en crudo o transformaciones de dichas señales.
2. **Geométrica**. Objetos en 2 o 3 dimensiones inferidos de los datos de los sensores.
3. **Relacional Local.** Relaciones funcionales, estructurales o semánticas entre objetos geométricos que se encuentran cerca uno del otro (ubicaciones individuales).
4. **Topológica**. Las ligas relacionales a gran escala que conectan objetos y ubicaciones a lo largo del medio ambiente como un todo.
5. **Semántica**. Etiquetas funcionales asociadas con los constituyentes del mapa.

2.7.1 Mapas Sensoriales

Los mapas basados en lecturas directas de los sensores ofrecen la posibilidad de acoplar la representación del medio ambiente lo más directo posible con los sensores que el robot utiliza para percibir el medio ambiente.

Entre los tipos más utilizados de mapas sensoriales es posible citar [Latombe 91]:

1. **Mapas basados en imágenes**.- Mientras un robot móvil se mueve dentro de un ambiente éste puede ir tomando lecturas con sus sensores. Si se asume una perfecta odometría, luego de cierto período habrá recabado cierto número de lecturas

$$I_i(x_i, y_i, \theta_i). \tag{2.6}$$

   Si el robot colecta suficientes lecturas de tal forma que se pueda calcular una aproximación continua de $I(x, y, \theta)$ entonces se podrían utilizar métodos de servo-control para navegar respecto a $I(x, y, \theta)$. La dificultad de este tipo de aproximación consiste en saber cómo muestrear el conjunto de posibles mediciones y cómo construir la gráfica continua I a partir de las mediciones individuales $I_i(x_i, y_i, \theta_i)$.

2. **Representaciones de Ocupación Espacial.**- Consiste en simplemente muestrear el espacio bidimensional o tridimensional y almacenar la ocupación de cada ubicación de muestreo individual. La forma más simple de realizar esto consiste en muestrear el espacio a lo largo de una malla en dos o tres dimensiones. En el caso de una representación bidimensional el mapa puede ser visto como un mapa de bits (bitmap) o píxeles (pixmap) si cada celda es vista como un píxel de imagen. El número de celdas requeridas para representar un ambiente utilizando una malla uniforme de ocupación es $O(n^d)$, donde n es el número de celdas utilizadas a lo largo de cada dimensión y d es el número de dimensiones del mapa.

   Una técnica común de ocupación espacial consiste en almacenar valores dentro de una malla de ocupación que refleje el grado de ocupación para cada celda: 1 para una celda completamente ocupada y 0 para una celda vacía. Otra alternativa consiste en asignar una cierta probabilidad $P_{ocupada}$ de que la celda esté ocupada. Una tercera variación consiste en almacenar la extensión que se encuentra ocupada de la celda: llena, parcialmente llena o completamente vacía.

2.7.2 Mapas Geométricos

Los mapas geométricos son elaborados a base de primitivas geométricas discretas: líneas, polígonos o poliedros, puntos, funciones polinomiales, etc. Estos mapas tienen la ventaja de ser altamente eficientes en cuanto a espacio de almacenamiento, debido a que una región arbitraria del espacio puede ser representada mediante un modelo con sólo unos pocos parámetros. Adicionalmente los mapas geométricos pueden almacenar datos de ocupación con casi arbitraria alta resolución sin ser susceptibles a errores de almacenamiento provocados por las técnicas basadas en el muestreo espacial.

Los mapas geométricos se caracterizan por dos aspectos clave:

- El conjunto de primitivas utilizado para describir los objetos
- El conjunto de operadores de composición y deformación

Aunque los sólidos platónicos son suficientes para tareas simples de navegación, un modelado más elaborado, como los métodos para recoger objetos o los problemas de reconocimiento basados en marcas (*landmarks*), han motivado al uso de primitivas más sofisticadas que incluyen las siguientes:

- Mapas en dos dimensiones:
  - Puntos
  - Líneas y segmentos de líneas
  - Círculos y arcos de círculos
  - Polinomios
  - Poliedros
  - Splines

- Mapas en tres dimensiones
  - Puntos
  - Superficies planas
  - Poliedros regulares, poliedros en general
  - Círculos y elipsoides
  - Supercuádricas
  - Superficies de producto tensorial, NURBS (Splines racionales no-uniformes) y superficies relacionadas con Splines

Mientras que los operadores básicos para manipular dichas primitivas incluyen:

- Transformaciones rígidas (rotación, escalamiento, translación)
- Transformaciones conformes (transformaciones que preservan la forma)
- Transformaciones afines
- Operaciones booleanas (geometría sólida constructiva: unión, intersección, etc.)
- Conjunto regularizado de operadores booleanos

El problema primario de las representaciones geométricas basadas en modelos es el hecho de que son difíciles de inferir a partir de los datos de los sensores. Regularmente se encuentran tres problemas fundamentales de modelado:

1. Falta de estabilidad.- La representación puede cambiar de manera drástica dada una mínima variación en la entrada.

2. Falta de unicidad.- Muchos ambientes diferentes pueden resultar en la misma representación.

3. Falta de poder expresivo.- Puede resultar difícil (o imposible) representar las protuberancias o salientes del ambiente dentro del sistema de modelado.

Estas dificultades surgen debido a que los parámetros individuales del modelo son difíciles de calcular con confiabilidad y especialmente debido a que, para una escena definida a partir de múltiples clases de modelos, puede resultar extremadamente difícil asociar modelos específicos que se adapten a conjuntos de mediciones específicos. La falta de estabilidad se refiere al hecho de que los modelos geométricos, generados como resultado de un conjunto particular de observaciones, pueden cambiar rápidamente con ligeras variaciones en los datos de entrada. La falta de unicidad o representación única ocurre debido a que muchos sistemas de modelado pueden expresar un conjunto simple de observaciones en más de una forma.

2.7.3 Mapas Topológicos

Las representaciones geométricas utilizan datos métricos como base de la representación. Desafortunadamente estos datos métricos en su mayoría pueden aparecer corruptos debido a ruido en los sensores. Para evitar confiar en datos con errores, una representación topológica no-métrica puede ser utilizada.

Las representaciones topológicas evitan los costos de almacenamiento potencialmente masivos asociados con las representaciones métricas. Adicionalmente muestran cierta similitud con percepción espacial humana. Las representaciones puramente topológicas, sin ninguna información de distancia, representan el peor escenario posible para los esquemas de posicionamiento basados en sensores.

La clave para una relación topológica consiste en alguna representación explícita de la conectividad entre regiones u objetos. En su forma más pura, esto puede involucrar una completa ausencia de datos métricos. Una representación topológica se basa en una abstracción del medio ambiente en términos de lugares discretos con aristas conectándolos; por ejemplo, un grafo

$$G = (V, E) \tag{2.7}$$

con un conjunto de N vértices V y un conjunto de M. aristas E.

Los vértices se denotan por:

$$V = \{v_1, ..., v_n\}$$

y las aristas por:

$$E = \{e_1, ..., e_m\}$$

y la arista e_{ij} está dada por:

$$e_{ij} = \{v_i, v_j\}$$

Si el orden de v_i y v_j es significante, como en los modelos que presuponen que las rutas son progresivas, entonces se tiene un *grafo dirigido*. Nótese que $M. < N\,(N–1)$ para este tipo de grafos sin aristas transitivas (aristas saliendo y entrando al mismo nodo), mientras que para *grafos planos* (grafos sin aristas que se crucen) se tiene como límite para el número de aristas $M. < 3N–1$.

A menudo sucede que el grafo *G* se encuentra contenido dentro de algún espacio métrico (los vértices tienen longitud y los vértices se encuentran orientados con respecto a los nodos).

El uso de grafos, y en particular grafos contenidos con aristas y vértices aumentados con diversas etiquetas, ha sido explotado por muchos sistemas robóticos para representar el medio ambiente. En el ejemplo siguiente extraído de [Krose 96] el medio ambiente del robot es modelado como un grafo cuyos vértices corresponden con marcas visuales (*landmarks*) ubicadas en el techo dentro del ambiente del robot. Cada marca es única y también define una orientación local. Las marcas son localizadas utilizando una cámara montada sobre el robot apuntando directamente hacia arriba. La representación de dicho ambiente se muestra en la Figura 2.5. Cada vértice corresponde con una de las marcas únicas, donde las aristas corresponden a caminos rectos entre las marcas. Cada arista en el grafo es etiquetada con la distancia que necesita ser recorrida a lo largo de dicha arista para llegar a la marca siguiente. Las aristas también son etiquetadas para mostrar su dirección con respecto a la orientación local definida por la marca.

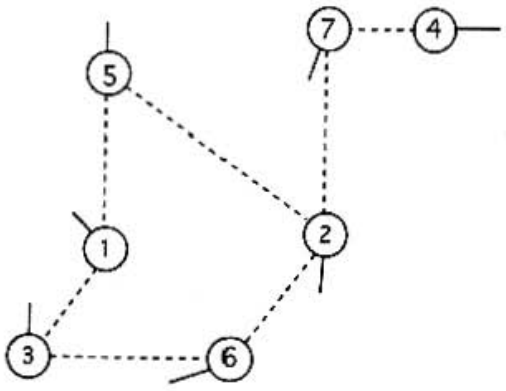

Figura 2.5: Representación del ambiente de operación de un robot basada en un grafo.
Los vértices corresponden a marcas conocidas y las aristas a la ruta entre ellos (extraído de [Krose 96]).

Capítulo 3

Redes Neuronales

3.1 Introducción

"Los modelos computacionales para las redes neuronales poseen una rica historia paralela al desarrollo de la inteligencia artificial simbólica tradicional. Algunos de los primeros trabajos en el área se refieren al modelo de neurona de McCulloch y Pitts (1943). McCulloch y Pitts utilizaron un modelo de neurona artificial con un umbral de activación y pesos sinápticos asociados a cada calor de entrada. Si el valor de activación era superado la neurona se disparaba llevando su salida a la siguiente neurona. Este simple modelo dio origen a redes capaces de aprender tareas simples de reconocimiento de patrones. Más tarde Rosenblatt (1958) introdujo un modelo formal de neurona llamado *perceptrón* y la prueba de convergencia asociada al perceptrón que estableció propiedades de aprendizaje comprobables para estos sistemas de red. En los años 60 y 70, sin embargo, la investigación en redes neuronales sufrió un declive por varias razones, incluyendo la publicación del libro *Perceptrones* (Minsky y Papert 1969), que comprobaron las limitaciones de las redes de perceptrones de una sola capa.

En los años 80 el área resurge con el advenimiento de las redes de perceptrones multicapa y el uso del algoritmo de retropropagación (Rumelhart, Hinton y Williams 1986) como medio para entrenar tales sistemas. Muchos otros esfuerzos notables durante la década pasada han aportado resultados altamente significativos. Debe recordarse que la mayoría de las redes neuronales han sido inspiradas sólo en neuronas biológicas reales y han tenido un desempeño muy pobre en funciones cerebrales. Por el contrario, estos modelos computacionales abstractos han tenido gran relevancia para los robots basados en comportamientos y han sido utilizados ampliamente en tareas que van desde el seguimiento de rutas mediante visión hasta la adaptación y aprendizaje en sistemas de control"

[Arkin 98].

3.1.1 Modelo General de Neurona Artificial (grupo PDP)** [Rumelhart 86]

Según [Brío 02] se define un modelo de neurona artificial como sigue:

Se denomina procesador elemental o neurona a un dispositivo simple de cálculo que, a partir de un vector de entrada procedente del exterior o de otras neuronas, proporciona una

única respuesta o salida. Los elementos que constituyen la neurona de etiqueta *i* son los siguientes:

- Conjunto de **entradas**, $x_j(t)$.
- **Pesos sinápticos** de la neurona *i*, w_{ij} que representan la intensidad de interacción entre cada neurona presináptica *j* y la neurona postsináptica *i*.
- **Regla de Propagación** $\sigma(w_{ij}, x_j(t))$, que proporciona el valor del potencial post-sináptico $h_i(t) = \sigma(w_{ij}, x_j(t))$ de la neurona *i* en función de sus pesos y entradas.
- **Función de Activación** $f_i(a_i(t-1), h_i(t))$, que proporciona el estado de activación actual $a_i(t) = f_i(a_i(t-1), h_i(t))$ de la neurona *i*, en función de su estado anterior $a_i(t-1)$ y de su potencial postsináptico actual.
- **Función de Salida** $F_i(a_i(t))$, que proporciona la salida actual $y_i(t) = F_i(a_i(t))$ de la neurona *i* en función de su estado de activación.

De este modo la operación de la neurona *i* puede expresarse como

$$y_i(t) = F_i(f_i[a_i(t-1), \sigma(w_{ij}, x_j(t))]) \tag{3.1}$$

Este modelo de neurona formal se inspira en la operación de la biológica, en el sentido de integrar una serie de entradas y proporcionar cierta respuesta, que se propaga por el axón.

3.1.2 Modelo Estándar de Neurona Artificial

El modelo anterior resulta muy general. En la práctica suele utilizarse uno más simple, denominado neurona estándar, considerando que la regla de propagación es la suma ponderada y que la función de salida es la identidad. De esta forma la **neurona estándar** consiste en:

- Un conjunto de **entradas**, $x_j(t)$ y pesos sinápticos w_{ij}.
- Una **regla de propagación** $h_i(t) = \sigma(w_{ij}, x_j(t))$; $h_i(t) = \Sigma\, w_{ij}\, x_j$ es la más común.
- Una **función de activación** $y_i(t) = f_i(h_i(t))$, que representa simultáneamente la salida de la neurona y su estado de activación.

Con frecuencia se añade al conjunto de pesos de la neurona un parámetro adicional θ_i, que se denomina **umbral**, que se resta del potencial postsináptico, por lo que el argumento de la función de activación queda

$$\Sigma\, w_{ij}\, x_j - \theta_i \tag{3.2}$$

lo que representa añadir un grado de libertad adicional a la neurona. En el caso de nodos de respuesta todo-nada este parámetro representará el umbral de disparo de la neurona, es decir, el nivel mínimo que debe alcanzar el potencial postsináptico (o potencial de membrana) para que la neurona se dispare o active.

En conclusión el modelo de neurona que se denomina estándar queda

$$y_i(t) = f_i\,(\Sigma\, w_{ij}\, x_j - \theta_i) \tag{3.3}$$

Ahora bien, si se hace que los índices *i* y *j* comiencen en 0, es posible definir $w_{i0} = \theta_i$ y $x_0 = -1$ (constante), con lo que el potencial postsináptico (potencial loca o de membrana) se obtiene realizando la suma desde *j*=0

$$y_i(t) = f_i\,(\sum_{j=0}^{n} w_{ij}\, x_j) \tag{3.4}$$

Definida de esta manera la neurona estándar, basta con establecer la forma de la función de activación para determinarla por completo.

3.1.3 Neuronas todo-nada (dispositivo de umbral)

Si en el modelo de neurona estándar se considera que las entradas son digitales, por ejemplo [0,1] y la función de activación es la escalón $H(\cdot)$ (denominada también de Heavside), definida entre 0 y 1 se tiene

$$y_i(t) = H\,(\sum_{j} w_{ij}\, x_j - \theta_i) \tag{3.5}$$

Como $H(x)=1$ cuando $x \geq 0$, y $H(x)= 0$ cuando $x<0$, se tiene

$$y_i(t) = \begin{cases} 1, \text{ si } \Sigma\, w_{ij}\, x_j \geq \theta_i \\ 0, \text{ si } \Sigma\, w_{ij}\, x_j < \theta_i \end{cases} \tag{3.6}$$

Es decir, si el potencial de membrana supera el valor de umbral θ_i (umbral de disparo), entonces la neurona se activa; si no lo supera, la neurona no se activa. Este es el modelo de neurona del perceptrón original, que se denomina en ocasiones dispositivo de tipo umbral.

3.1.4 Neurona continua sigmoidea

Si en el esquema de neurona estándar se considera que las entradas pueden ser tanto digitales como continuas (analógicas), y las salidas exclusivamente continuas, puede emplearse como función de activación una sigmoidea (cuya gráfica tiene forma de "S" inclinada y aplastada) que es una función continua y diferenciable en cierto intervalo, por ejemplo en el [−1,+1] o en el [0, +1], dependiendo de la función concreta elegida. Las dos funciones más habituales de este tipo son:

$$y = f(x) = \frac{1}{1 + e^{-x}}, \text{ con } y \in [0,1] \tag{3.7}$$

$$y = f(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}, \text{ con } y \in [-1,1] \tag{3.8}$$

Este modelo de neurona es el utilizado en el perceptrón multicapa. El requisito de trabajar con funciones diferenciables lo puede imponer la regla de aprendizaje, como sucede con la famosa retropropagación (bp ó backpropagation).

3.1.5 Arquitectura de una red neuronal

Se denomina arquitectura a la topología, estructura o patrón de conexiones de una red neuronal; esta estructura de conexiones determina el comportamiento de la red. Las conexiones entre neuronas son progresivas, es decir, la información sólo puede propagarse en un solo sentido. En general las neuronas se suelen agrupar en **capas**, las capas en grupos (clusters) y a todo el conjunto de capas se denomina ***red neuronal***.

Es posible distinguir tres tipos de capas: de entrada, de salida y ocultas. Una capa de entrada está compuesta por neuronas que reciben los datos directamente del entorno, una capa de salida proporciona la respuesta de la red neuronal mientras que una capa oculta no tiene conexión directa con el entorno.

Atendiendo a distintos conceptos, se establecen varios tipos de arquitecturas neuronales, de tal forma que, en cuanto a su estructura en capas, es posible hablar de arquitecturas monocapa (una sola capa de neuronas) y "redes" o arquitecturas multicapa (varias capas).

En cuanto al flujo de datos dentro de la red neuronal, se habla de redes progresivas (*feed-forward*) y redes retroalimentadas (*feedback*). En las redes progresivas, la información circula en un único sentido, desde las neuronas de entrada hacia las de salida. En las redes retroalimentadas la información puede circular entre las capas en cualquier sentido, incluso el de salida-entrada.

3.1.6 Aprendizaje en una red neuronal

En el contexto de las redes neuronales se define el aprendizaje como el proceso por el que se produce el ajuste de los parámetros libres de la red a partir de un proceso de estimulación por el entorno que rodea la red. En la mayor parte de las ocasiones el aprendizaje consiste en simplemente determinar un conjunto de pesos sinápticos que permita a la red realizar correctamente el tipo de procesamiento deseado.

Cuando se construye un sistema neuronal, se parte de un cierto modelo de neurona y de una determinada arquitectura de red, estableciéndose los pesos iniciales nulos o aleatorios. Para que la red resulte operativa es necesario entrenarla, lo que constituye el ***modo aprendizaje***. Una vez entrenada la red neuronal es posible pasar a ***modo recuerdo*** alimentando sus entradas y observando la salida proporcionada por la red. El entrenamiento o aprendizaje se puede llevar a cabo a dos niveles. El más convencional es el de modelado de las sinapsis, que consiste en modificar los pesos sinápticos siguiendo una cierta regla aprendizaje, construida normalmente a partir de la optimización de una función de error o costo, que mide la eficacia actual de la operación de la red. Si se denomina $w_{ij}(t)$ al peso que conecta la neurona presináptica j con la postsináptica i en la iteración t, el algoritmo de aprendizaje, en función de las señales que en el instante t llegan procedentes del entorno,

proporcionará el valor $\Delta w_{ij}(t)$ que da la modificación que se debe incorporar en dicho peso, el cual quedará actualizado de la forma

$$w_{ij}(t+1) = w_{ij}(t) + \Delta w_{ij}(t) \tag{3.9}$$

El proceso de aprendizaje es usualmente iterativo, actualizándose los pesos de la manera anterior, una y otra vez, hasta que la red neuronal alcanza el rendimiento deseado.

Algunos modelos neuronales incluyen otro nivel de aprendizaje: la creación o destrucción de neuronas, en el cual se modifica la propia arquitectura de la red. En cualquier caso, en un proceso de aprendizaje la información contenida en los datos de entrada queda incorporada en la propia estructura de la red neuronal, la cual almacena la representación de una cierta imagen de su entorno.

Los dos tipos básicos de aprendizaje son el supervisado y el no supervisado.

a) ***Aprendizaje supervisado:*** En el aprendizaje supervisado se presenta a la red un conjunto de patrones, junto con la salida deseada u objetivo, e iterativamente ésta ajusta sus pesos hasta que su salida tiende a ser la deseada, utilizando para ello información detallada del error que comete en cada paso. De este modo la red es capaz de estimar relaciones entrada/salida sin necesidad de proponer una cierta forma funcional de partida.

b) ***Aprendizaje no supervisado o autoorganizado:*** En el aprendizaje no supervisado se presentan a la red multitud de patrones sin adjuntar la respuesta deseada. La red, por medio de la regla de aprendizaje, estima la función de densidad de probabilidad $\mathbf{p}(\boldsymbol{x})$ (que describe la distribución de patrones x pertenecientes al espacio de entrada de dimensión n) a partir de muestras, con lo cual pueden reconocerse regularidades en el conjunto de entradas, extraer rasgos o agrupar patrones según su similitud (clustering).

c) ***Aprendizaje híbrido:*** En el aprendizaje híbrido coexisten en la red los dos tipos básicos de aprendizaje, el supervisado y el no supervisado, los cuales tienen lugar normalmente en distintas capas de neuronas.

d) ***Aprendizaje reforzado:*** El aprendizaje reforzado se sitúa a medio camino entre el supervisado y el autoorganizado. Como en el primero de ellos, se emplea información sobre el error cometido, pero en este caso existe una única señal de error que representa un índice global del rendimiento de la red (solamente le es indicado lo bien o mal que está actuando, pero sin proporcionar más detalles). Como en el caso del no supervisado, no se suministra explícitamente la salida deseada. En ocasiones se denomina aprendizaje por premio-castigo.

3.1.7 Fase de recuerdo

Generalmente (aunque no en todos los casos), una vez que el sistema ha sido entrenado, el aprendizaje se “desconecta”, por lo que los pesos y la estructura quedan fijos, estando la red neuronal ya dispuesta para procesar datos. Este modo de operación se denomina modo recuerdo (*recall*) o de ejecución.

En las redes progresivas, ante un patrón de entrada, las neuronas responden proporcionando directamente la salida del sistema. Al no existir realimentación, no existe ningún problema en relación con su estabilidad. Por el contrario, las redes con realimentación son sistemas dinámicos no-lineales, que requieren ciertas condiciones para que su respuesta converja a un estado estable o punto fijo.

3.2 Redes Progresivas

Como se mencionó anteriormente en las redes progresivas la información fluye en un solo sentido, desde las neuronas de entrada hacia las neuronas de salida, es decir, no se tiene realimentación entre neuronas. Dentro de este tipo de modelos de redes neuronales se cita el caso del perceptrón simple [Rosenblatt 62], el *adaline* y el perceptrón multicapa o MLP (Multilayer Perceptron) [Brío 02].

La historia de las redes neuronales se encuentra íntimamente ligada a dichos modelos. El perceptrón simple y el *adaline* fueron propuestos a finales de los años cincuenta y gozaron de gran popularidad. Sin embargo debido, al trabajo de Minsky y Paper [Minsky 69] el desarrollo en el campo se detuvo casi por completo debido a que dichos autores demostraron las limitaciones de los primeros modelos. No obstante en los años ochenta el tema resurgió debido al uso generalizado de computadoras con capacidades VLSI (Very Large Scale Integration) que permitían realizar cálculos y simulaciones más complejas. Es en estos años cuando el grupo PDP implementó el MLP entrenado mediante el algoritmo de retropropagación (*backpropagation*) que superaba las limitaciones de los modelos anteriores.

3.2.1 El Perceptrón Simple

El perceptrón simple fue introducido por Rosenblatt a principios de los años sesenta [Rosenblatt 62]. Es un modelo progresivo, compuesto por dos capas de neuronas, una sensorial o de entradas y otra de salida (Figura 3.1).

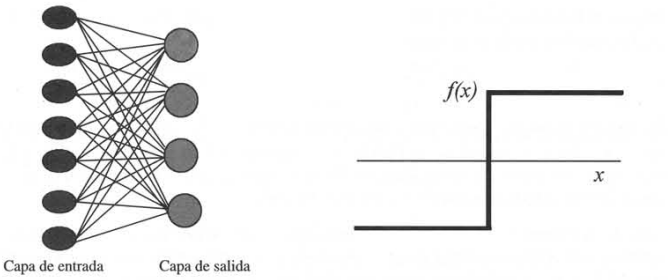

Figura 3.1: Perceptrón simple y función de transferencia de su neurona (tomada de [Brío 02]).

Las neuronas de entrada no realizan ningún cómputo, únicamente envían la información que, en principio, se considera como señales discretas [0, +1]. La operación de este tipo de red con *n* neuronas de entrada y *m.* de salida se puede expresar como:

$$y_i(t) = H\left(\sum_{j=1}^{n} w_{ij}\, x_j - \theta_i\right),\ 1 \le i \le m. \qquad (3.10)$$

donde H es la función escalón unitario.

El perceptrón puede utilizarse tanto como clasificador como para la representación de funciones booleanas. Su importancia radica en el hecho de que es un dispositivo entrenable ya que el algoritmo de aprendizaje introducido por Rosenblatt permite determinar automáticamente los pesos sinápticos que clasifican un conjunto de patrones a partir de un conjunto de pesos etiquetados. Cada neurona del perceptrón representa una determinada clase, de modo que, dado un vector de entrada, una cierta neurona responde con 0 si no pertenece a la clase que representa, y con un 1 si pertenece. Se puede observar que una neurona de este tipo solo permite discernir entre dos clases linealmente separables.

El algoritmo de aprendizaje del perceptrón [Brío 02] es un algoritmo de aprendizaje por corrección de errores donde se ajustan los pesos en proporción a la diferencia existente entre la salida actual de la red y la salida deseada, con el objetivo de minimizar el error actual de la red.

3.2.2 El Adaline

El *adaline* fue introducido por Widrow en 1960 [Widrow 60], cuyo nombre proviene de *ADAptive Linear Network*. Utiliza una neurona similar a la del perceptrón, pero de respuesta lineal cuyas entradas pueden ser continuas. Incorpora un parámetro adicional llamado *bias* o umbral, que proporciona un grado de libertad adicional.

La ecuación del *adaline* para n neuronas de entrada y m. de salida es:

$$y_i(t) = \sum_{j=1}^{n} w_{ij}\, x_j - \theta_i,\ \ 1 \le i \le m. \qquad (3.11)$$

La diferencia más importante con el perceptrón reside en la regla de aprendizaje que implementa. En el *adaline* se utiliza la regla de Widrow−Hoff, también conocida como regla LMS (*Least Mean Squares* o mínimos cuadrados) [Brío 02], que conduce a actualizaciones de tipo continuo, con la actualización de los pesos proporcional al error cometido por la neurona.

3.2.3 El Perceptrón Multicapa

Si se añade una o más capas al modelo del perceptrón simple, se obtiene un perceptrón multicapa o MLP. Esta arquitectura suele entrenarse mediante el algoritmo denominado retropropagación de errores o BP cuyo éxito se debe al trabajo del grupo PDP que lo presentaron a la comunidad internacional como una técnica útil de resolución de problemas complejos [Rumelhart 86].

En la Figura 3.2 se muestra la estructura del MLP. Se llama x_i a las entradas de la red, y_i a las salidas de la capa oculta y z_k a las de la capa final (y globales de la red); t_k son las salidas objetivo, w_{ji} son los pesos de la capa oculta y θ_j sus umbrales, w'_{kj} los pesos de la capa de salida y θ'_k sus umbrales. La operación de un MLP con una capa oculta y neuronas de salida lineal se puede expresar matemáticamente de la siguiente forma:

$$z_k = \sum_j w'_{kj}\, y_j - \theta'_k = \sum_j w'_{kj}\, f\left(\sum_i w_{ji}\, x_i - \theta_j\right) - \theta'_k \qquad (3.12)$$

siendo $f(\cdot)$ de tipo sigmoideo (ver sección 3.1.4).

3.2.4 Aprendizaje por retropropagación de errores (*Back Propagation o BP*)

Sea un MLP de tres capas cuya arquitectura se presenta en la Figura 3.3, con las entradas, salidas, pesos y umbrales de las neuronas definidas en la sección anterior.

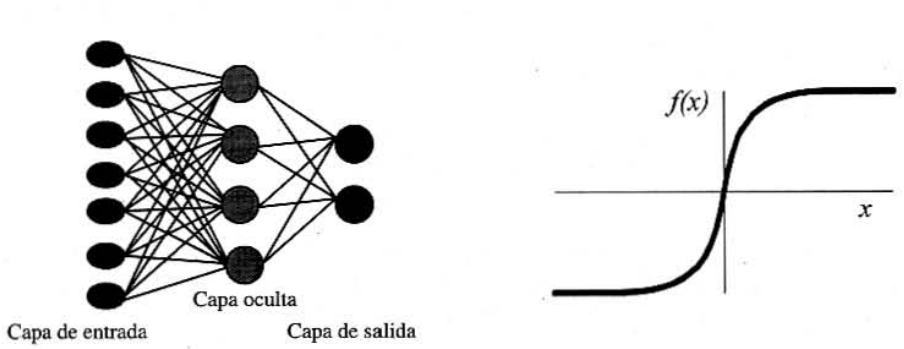

Figura 3.2: Perceptrón multicapa y función de transferencia de su neurona (tomada de [Brío 02]).

Dado un patrón de entrada x^μ, (μ=1,...,p), se tiene que

$$z^\mu_k = \sum_j w'_{kj}\, y^\mu_j - \theta'_k = \sum_j w'_{kj}\, f\left(\sum_i w_{ji}\, x^\mu_i - \theta_j\right) - \theta'_k \qquad (3.13)$$

siendo f la función de activación de las neuronas de la capa oculta de tipo sigmoideo, típicamente la ecuación (3.7).

Como en el caso del adaline, se determina una función de costo la cual se tratará de minimizar, en este caso se trata del error cuadrático medio

$$E\,(w_{ji},\, \theta_j\,,\, w'_{kj},\, \theta'_k\,) = (1/2) \left[\sum_\mu \sum_k \; t^\mu_k - f\left(\sum_j w'_{kj}\, y^\mu_j - \theta'_k\right) \right]^2 \qquad (3.14)$$

La minimización se lleva a cabo mediante descenso por el gradiente, pero en este caso existen dos gradientes, uno con respecto de la capa de salida y otro respecto a la capa de entrada. La actualización de los pesos según (2.3) puede expresarse como

$$\Delta w'_{kj} = -\varepsilon \frac{\partial E}{\partial w'_{kj}} \qquad \Delta w_{ji} = -\varepsilon \frac{\partial E}{\partial w_{ji}} \tag{3.15}$$

siendo ε conocido como ritmo o factor de aprendizaje (generalmente $0 < \varepsilon < 1$).

Lo anterior indica que se hará un ajuste en los pesos de la red en sentido contrario al gradiente (máxima tasa de aumento), es decir, en el sentido del máximo decremento de la función de costo. Las expresiones de actualización se obtienen sólo con derivar, teniendo en cuenta las dependencias funcionales y aplicando adecuadamente la regla de la cadena:

$$\Delta w'_{kj} = \varepsilon \sum_{\mu} \Delta'^{\mu}_{k} y^{\mu}_{j}, \text{ con } \Delta'^{\mu}_{k} = [t^{\mu}_{k} - f(v'^{\mu}_{k})] \frac{\partial f(v'^{\mu}_{k})}{\partial v'^{\mu}_{k}} \tag{3.16}$$

$$\Delta w_{ji} = \varepsilon \sum_{\mu} \Delta^{\mu}_{j} x^{\mu}_{i}, \text{ con } \Delta^{\mu}_{j} = \left(\sum_{k} \Delta'^{\mu}_{k} w'_{kj} \right) \frac{\partial f(v^{\mu}_{j})}{\partial v^{\mu}_{j}} \tag{3.17}$$

La actualización de los umbrales (bias) se realiza haciendo uso de estas mismas expresiones, considerando que el umbral es un caso particular de peso sináptico, cuya entrada es un valor constante igual a –1.

En estas expresiones está implícito el concepto de propagación hacia atrás de los errores que da nombre al algoritmo. En primer lugar se calcula la expresión Δ'^{μ}_{k} (3.16), que se denomina señal de error, por ser proporcional al error de salida actual de la red, con el que calculamos la actualización $\Delta w'_{kj}$ de los pesos de la capa de salida. A continuación se propagan hacia atrás los errores Δ'^{μ}_{k} a través de las sinapsis, proporcionando así las señales de error Δ^{μ}_{j} (3.17), correspondientes a las sinapsis de la capa oculta; con éstas se calcula la actualización Δw_{ij} de las sinapsis ocultas. El algoritmo puede extenderse fácilmente a arquitecturas con más de una capa oculta siguiendo el mismo esquema.

En el esquema presentado, que surge de forma natural del proceso de descenso por el gradiente, se lleva a cabo una fase de ejecución para todos los patrones del conjunto de entrenamiento, se calcula la variación de los pesos debida a cada patrón, se acumulan, y solamente entonces se procede a la actualización de los pesos. Este esquema se denomina **aprendizaje por lotes** (*batch*). Una variación al algoritmo consiste en actualizar los pesos sinápticos tras la presentación de cada patrón, esquema denominado **aprendizaje en tiempo real** (*on line*) que evita tener que presentar todos los patrones para poder actualizar los pesos sinápticos. Lo anterior resulta sumamente útil cuando se dispone de un elevado número de patrones de entrenamiento. Cabe destacar que se hace imprescindible presentar los patrones en orden aleatorio cada vez, con la finalidad de evitar que el entrenamiento se vicie en favor del último patrón presentado cuya actualización, por ser siempre la última, predominaría sobre las anteriores.

3.2.5 Aceleración del aprendizaje *BP*

Para tratar de resolver el problema de la lenta convergencia del BP los inventores del modelo [Rumelhart 86] propusieron una variante que consiste en añadir al cálculo de los pesos (3.16) y (3.17) un término adicional denominado **momento** proporcional al incremento de la iteración anterior (inercia).

$$\Delta w'_{kj}(t+1) = -\varepsilon \left. \frac{\partial E}{\partial w'_{kj}} \right|_t + \alpha \, \Delta w'_{kj}(t-1) \tag{3.18}$$

$$\Delta w_{ji}(t+1) = -\varepsilon \left. \frac{\partial E}{\partial w_{ji}} \right|_t + \alpha \, \Delta w_{ji}(t-1) \tag{3.19}$$

3.3 Redes de Agrupamiento (*Clustering*)

3.3.1 Redes Autoorganizadas

Las redes autoorganizadas son redes que se caracterizan porque en su entrenamiento no se presentan las salidas objetivo que se desean asociar a cada patrón de entrada. La red, a partir de un proceso de autoorganización, proporcionará cierto resultado, el cual será reflejo de las relaciones de similitud existentes entre dichos patrones de entrada. La principal aplicación de estos modelos será la realización de agrupamiento de patrones (*clustering*), análisis exploratorio y visualización y minería de datos (*data mining*).

3.3.2 Modelos Neuronales No Supervisados

En los modelos de aprendizaje no supervisado no existe ninguna guía que le indique a la red si está operando correcta o incorrectamente, pues no se dispone de ninguna salida objetivo hacia la cual la red neuronal deba tender. Así, durante el proceso de aprendizaje la red debe descubrir por sí misma rasgos comunes, regularidades, correlaciones o categorías en los datos de entrada, e incorporarlos a su estructura interna de conexiones (pesos). Para obtener resultados de calidad la red requiere de un cierto nivel de redundancia en las entradas procedentes del espacio sensorial.

Existen diversas arquitecturas de redes neuronales no supervisadas que realizan diferentes funciones [Hertz 91].

a) Análisis de similitud entre patrones (o novedad). El procesamiento por similitud entre patrones aparece cuando existe una única neurona cuya salida es continua, indicando el grado de similitud o parecido entre el patrón de entrada actual y el promedio de los presentados en el pasado. Dicho promedio queda representado durante el entrenamiento en el vector de pesos sinápticos de la neurona, de modo que la neurona aprende lo que es típico dentro de un conjunto de patrones.

b) Análisis de componentes principales. Si se extiende el caso anterior a varias neuronas de salida continua, el proceso de aprendizaje supone encontrar una cierta base del espacio de entrada, representada por el conjunto de los vectores de pesos sinápticos de todas las neuronas, que se corresponderán con los rasgos más sobresalientes del espacio sensorial.

c) Agrupamiento/clasificación (clustering). En este tipo de clasificación la red realiza tareas de agrupamiento cuando se compone de neuronas de salida discreta [0,1], donde cada una representa una categoría, y solamente una de ellas puede permanecer activada a la vez. Ante un patrón de entrada, la neurona que se activa indica a qué categoría o grupo (cluster) pertenece. Durante el aprendizaje la red deduce las categorías presentes en el espacio de entrada a partir de la medida de distancias (euclideana o de otro tipo) entre los patrones presentados.

d) Memoria asociativa. Este tipo de red representa una generalización del caso anterior de redes para agrupamiento: en este caso la salida proporcionada por la neurona activada no es solamente un 0 o un 1, sino el vector prototipo de la clase en cuestión.

e) Codificación: Es análogo al anterior, sólo que en este caso la neurona no proporciona como salida el vector prototipo de la clase, sino una versión codificada (por ejemplo, una etiqueta), habitualmente empleando menos bits y manteniendo la información relevante (compresión de datos).

f) Mapas de rasgos. Los mapas de rasgos son modelos no supervisados en los que las neuronas se ordenan geométricamente (por ejemplo, en forma de matriz bidimensional o mapa), llevando a cabo una proyección del espacio sensorial de entrada sobre la red (mapa), proyección que preserva en lo posible la topología del espacio original, pero reduciendo sus dimensiones.

Los casos citados no son necesariamente diferentes ni excluyentes; un mismo modelo de red no supervisada puede estar capacitado para efectuar varios de estos procesamientos. Por ejemplo, la codificación puede realizarse mediante un análisis de componentes principales o mediante *clustering*. Este último caso recibe el nombre de **cuantificación vectorial** (*vector quantization*).

3.3.3 Aprendizaje Competitivo en Redes de Agrupamiento

De acuerdo con [Krose 96] el aprendizaje competitivo es un procedimiento de aprendizaje que divide un conjunto de patrones de entrada en grupos o clusters que son inherentes a los datos de entrada. Una red de aprendizaje competitivo es provista únicamente con vectores de entrada x y con ellos implementa un procedimiento de aprendizaje no supervisado.

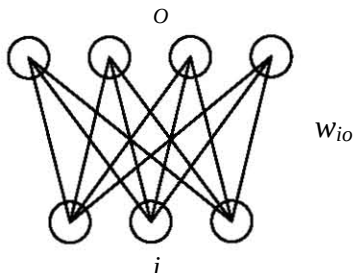

Figura 3.3: Una red competitiva simple.
Cada una de las salidas *o* está conectada a todas las entradas *i*.

Un ejemplo de red de aprendizaje competitivo se muestra en la Figura 3.3. Todas las neuronas de salida *o* se encuentran conectadas con todas las neuronas de entrada *i* con pesos sinápticos w_{io}. Cuando un patrón de entrada *x* se presenta a la red, únicamente una neurona de salida (la ganadora) se activa. En una red correctamente entrenada, todos los patrones *x* dentro del mismo grupo o cluster tendrán la misma ganadora. Para determinar la neurona ganadora y la regla de aprendizaje correspondiente existen dos métodos:

a) Selección de la ganadora mediante producto punto

Para poder utilizar este método se debe asumir que tanto los vectores de entrada *x* como los vectores de pesos w_o se encuentran normalizados a la unidad. Cada neurona de salida *o* calcula su valor de activación y_o de acuerdo con el producto punto de la entrada y el vector de pesos de dicha neurona

$$y_o = \sum_i w_{io}\, x_i = w^T{}_o\, x \tag{3.20}$$

En el paso siguiente, le neurona de salida *k* es seleccionada con la máxima activación

$$\forall o \neq k: \qquad y_o \leq y_k \tag{3.21}$$

Las activaciones entonces se fijan de tal forma que $y_k = 1$ y $y_{o\neq k} = 0$. Esto constituye el aspecto competitivo de la red, y la capa de salida es referida como capa "el ganador se lleva todo". Esta capa usualmente se implementa en software simplemente seleccionando la neurona de salida con el valor más alto de activación.

Una vez que el ganador *k* se ha seleccionado, los pesos se actualizan de acuerdo a

$$w_k(t+1) = \frac{w_k(t) + \gamma\big(x(t) - w_k(t)\big)}{\big\| w_k(t) + \gamma\big(x(t) - w_k(t)\big) \big\|} \tag{3.22}$$

donde γ es el ritmo de aprendizaje y el divisor asegura que todos los vectores de pesos *w* estén normalizados. Nótese que sólo se actualizan los pesos de la neurona ganadora.

La actualización de los pesos con la ecuación (3.20) efectivamente rota el vector w_o hacia el vector de entrada *x*. Cada vez que una entrada *x* se presenta, el vector de pesos más cercano a esta entrada se selecciona y es subsecuentemente rotado hacia la entrada (Fig. 3.4). Consecuentemente los vectores de pesos se rotan hacia aquellas áreas donde aparecen muchas entradas: los grupos o clusters de entradas.

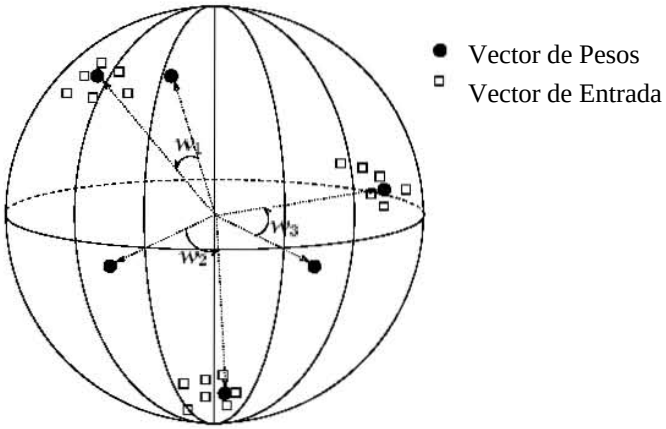

Figura 3.4: Ejemplo de agrupamiento en tres dimensiones con vectores normalizados (que caen dentro de la esfera unitaria). Los tres vectores de pesos se rotan hacia los centros de gravedad de los tres diferentes grupos o clusters de entrada (tomado de [Krose 96]).

b) Selección de la ganadora mediante distancia euclidiana

Previamente se asumió que tanto las entradas *x* como los vectores de pesos *w* se encuentran normalizados. Por esta razón, el algoritmo falla si se utilizan vectores sin normalizar. Naturalmente se desearía adaptar el algoritmo para poder utilizar vectores sin normalizar. Para que esto suceda la neurona *k* se selecciona con su vector de pesos w_k más cercano al patrón de entrada *x*, utilizando una medida de distancia euclideana:

$$k \text{ tal que} \left\| w_k - x \right\| \leq \left\| w_o - x \right\| \forall o \tag{3.23}$$

La distancia euclideana constituye un caso más general de las ecuaciones (3.20) y (3.21). En lugar de rotar el vector de pesos hacia la entrada con la ecuación (3.22), la actualización de los pesos debe ser modificada para implementar un *corrimiento* hacia la entrada:

$$w_k(t+1) = w_k(t) + \gamma\big(x(t) - w_k(t)\big) \tag{3.24}$$

De nuevo únicamente se actualizan los pesos de la neurona ganadora.

Un punto a tener en cuenta consiste en la inicialización de los vectores de pesos. Especialmente si los vectores de entrada se toman de un gran espacio de entrada multi-dimensional, es fácil de imaginar que un vector de pesos w_o generado al azar nunca será elegido como el ganador y por lo tanto nunca será modificado y usado. Por lo tanto, es

común inicializar cada vector de pesos a partir de un vector de entrada elegido al azar. Otro método se denomina aprendizaje de fuga (*leaky learning*). Se implementa expandiendo la ecuación (3.24) como sigue:

$$w_l(t+1) = w_l(t) + \gamma'\big(x(t) - w_l(t)\big) \qquad \forall l \neq k \tag{3.25}$$

con $\gamma' << \gamma$ llamado el factor de aprendizaje de fuga (*leaky learning rate*). Lo anterior hace que el resto de los vectores de pesos no seleccionados se acerquen hacia la entrada ligeramente, lo que aumentará la probabilidad de ser elegidos en alguna iteración posterior para algún vector de entrada *x*.

3.4 Redes de Hopfield

3.4.1 Descripcion

En la literatura existen distintos modelos de redes realimentadas, aunque el denominado red de Hopfield [Hopfield 82] quizás sea el más conocido. En 1982 Hopfield conjuntó diversas ideas sobre este tipo de redes y presentó un análisis matemático completo basado en modelos de espín (el giro o espin de un electrón puede ser –1/2 o 1/2), de tipo Ising [Amit 86].

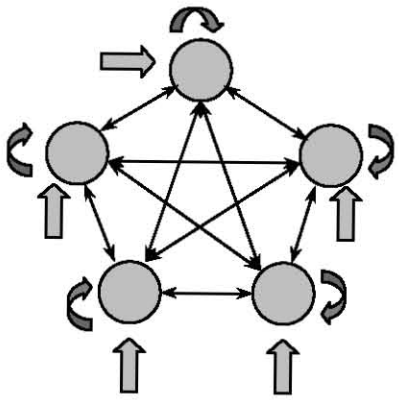

Figura 3.5: La red de autoasociación. Todas las neuronas son tanto de entrada como de salida. Cuando un patrón es presentado, la red oscila hasta alcanzar un estado estable y la salida de la red consiste en el nuevo estado de activación de todas las neuronas.

La arquitectura de la red de Hopfield consiste básicamente en una única capa de neuronas, donde cada una se conecta con todas las demás (Fig. 3.5). El modelo de neurona empleado tiene una estructura similar a la del perceptrón, un sencillo elemento de umbral, con entradas $x_j(t)$ binarias ([0,1] ó [–1,+1], según convenga), y salidas que en principio se llamarán $y_i(t)$, también binarias. Los pesos w_{ij} son continuos, es decir, números reales. La neurona *i* calcula el potencial postsináptico h_i (también denominado potencial local o de membrana), como la suma de las entradas ponderada con los pesos sinápticos, menos un cierto umbral de disparo.

$$h_i(t) = \sum_j w_{ij}\, x_j(t) - \theta_i \tag{3.26}$$

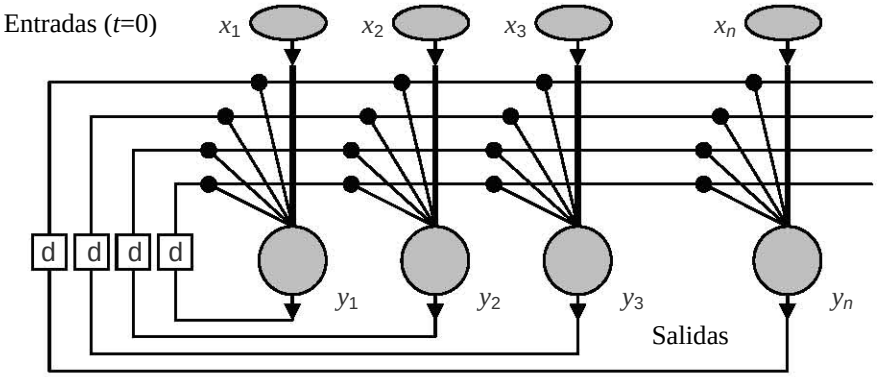

Figura 3.6: Arquitectura de la red de Hopfield. La letra ´d´ representa un retraso de una iteración (una salida en tiempo *t* se convierte en una entrada *x* al tiempo *t*+1).

Finalmente, al potencial local se aplica una función de tipo escalón $f(\cdot)$ para obtener la salida digital de la neurona

$$y_i(t) = f(h_i(t)) = f\left(\sum_j w_{ij}\, x_j(t) - \theta_i\right) \tag{3.27}$$

Cuando las neuronas hacen uso de valores [–1,+1] se denominan de tipo **Ising** [Muller 90], siendo la función de activación que emplea la signo-de-equis, $f(x)=\text{signo}(x)$. En este caso, el estado +1 indica una neurona activada, y el –1 desactivada.

En un primer estado $t=0$, las neuronas reciben las entradas provenientes del exterior $x_j(0)$, calculando las salidas asociadas $y_i(0)$, Debido a las realimentaciones (Figura 3.6), estas salidas se convierten en las nuevas entradas de la red $x_j(1)$, proporcionando una nueva salida $y_i(1)$. En general, ante las entradas $x_i(t)$, la red proporciona una salida $y_i(t)$, que se convierten en las nuevas entradas $x_i(t+1)$, de modo que la operación de la red de Hopfield se puede expresar como

$$x_i(t+1) = f\left(\sum_j w_{ij}\, x_j(t) - \theta_i\right) \tag{3.28}$$

regla que rige su dinámica. En resumen, al comienzo ($t=0$) cada neurona recibe del exterior un vector de entradas $x_j(0)$, de modo que cada neurona *i* queda situada inicialmente en este estado, cuyo valor será 0 o 1 (o bien, –1 o +1, según nuestra elección). A partir de entonces la red neuronal interrumpe su comunicación con el exterior y procesa las entradas recibidas; para tiempos $t+1$ sucesivos, las entradas de cada neurona son el estado en el que se sitúan en el instante anterior *t*. Habitualmente se dice que el estado $\boldsymbol{x}$ de la red de Hopfield en *t* viene determinado por el estado de activación de todas y cada una de sus neuronas.

$$\mathbf{x}(t) = \left(x_1(t)...x_i(t)...x_n(t)\right)^{\mathrm{T}} \tag{3.29}$$

En general, una red de Hopfield, en cada iteración *t*, pasa de un estado ***x*(*t*)** a otro estado ***x*(*t*+1)**. El proceso finalizará cuando se alcance un estado estable o punto fijo de la red ***x****, es decir, un estado que cumpla la siguiente condición a partir de un cierto *t*

$$x(t+1) = x(t) \equiv x^* \tag{3.30}$$

pues ello supondrá que la salida de la red ya no cambia, es decir, que la red se ha estabilizado. En ese momento se puede suponer que la red neuronal ha acabado de procesar el patrón original procedente del exterior ***x***(0), siendo ***x**** la respuesta final que proporciona.

3.4.2 Energía de la Red

El comportamiento del sistema puede ser descrito mediante una función de energía de la red [Brío 02]

$$E = -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij}x_i x_j + \sum_{i=1}^{n}\theta x_i \tag{3.31}$$

Por construcción, la variación de esta energía es siempre menor o igual a cero, es decir, en la operación de la red (ecuación 3.28) su energía nunca crece. Como la energía está limitada, la red siempre llegará a un estado de mínima energía, que será un mínimo local de la energía de la red. Este mínimo local corresponderá con un estado estable, punto fijo o atractor.

Realizando un estudio riguroso se llega a la conclusión que para que la ecuación (3.31) sea no creciente, bastan las siguientes dos condiciones

$$w_{ij} = w_{ji} \quad \text{y} \quad 1 > w_{ii} \geq 0$$

3.4.3 Dinámicas de la Red

Es posible plantear dos esquemas de actualización para una red de Hopfield [Muller 90]

a) Dinámica asíncrona o modo serie de operación, también denominada de Glauber. En un instante *t* solamente una neurona de la red actualiza su estado. Ésta puede ser seleccionada aleatoriamente, o bien siguiendo un cierto orden preestablecido.

b) Dinámica síncrona o modo paralelo de operación, también denominada de Little. En un instante *t* varias neuronas actualizan su estado a la vez. En el caso en que todas las neuronas de la red lo hagan al unísono se tiene el modo **completamente en paralelo**.

Distintas dinámicas aplicadas sobre una misma arquitectura de red de Hopfield hacen que opere de manera diferente y, por lo tanto, que el estado final de las neuronas sea también distinto [Bruck 90].

3.4.3 Memoria Asociativa

A diferencia de lo que sucede en la memoria de las computadoras, en la memoria asociativa la información no se encuentra almacenada en direcciones fijas o "casillas" sino más bien aparece dispersa (distribuida) como sucede en el cerebro humano. Para recuperarla no es posible acceder a una dirección determinada de la memoria, sino que es recuperada a partir de alguna pista, información parcial o por contexto.

Las redes neuronales operan siguiendo este mismo esquema. Por ejemplo una red de perceptrones implementa una memoria asociativa de tipo **heteroasociativo**, al asociar un patrón de entrada a cierto patrón de salida diferente, en general. Por el contrario, el modelo de Hopfield toma el papel de una memoria de tipo **autoasociativo**, pues, en definitiva, está ideada para asociar un patrón de entrada a sí mismo. En este caso los pesos de las conexiones entre las neuronas deben fijarse de tal forma que los estados del sistema que corresponden a los patrones que van a ser almacenados sean estables. Estos estados pueden verse como "cuencas" en el espacio de la energía de la red. Cuando la red se alimenta con un patrón con ruido o incompleto, la red completa la información faltante o errónea iterando hasta algún estado estable "cercano" al patrón presentado.

3.4.4 Aprendizaje en la Red de Hopfield

Vista la red de Hopfield como memoria asociativa, un estado estable o atractor de la red es un mínimo local de la función energía, por lo tanto, los recuerdos o memorias que la red almacena son mínimos locales de la red, por lo que el aprendizaje consistirá en conseguir que la red almacene (o memorice) como estados estables un conjunto de recuerdos o memorias dado.

3.4.4.1 Regla de Hebb

La primera regla propuesta para este modelo de red fue la regla de Hebb, que el propio Hopfield sugirió en su trabajo. Esta regla supone una neurona tipo Ising [−1,1] y umbrales nulos, y sea un conjunto de p patrones x^{μ}, $\mu=1,...,p$, que se desean memorizar. La regla de Hebb en este caso se expresa

$$w_{ij} = \frac{1}{n}\sum_{\mu=1}^{p} x_i^{\mu}\, x_j^{\mu} \tag{3.32}$$

que como puede observarse, cumple las dos condiciones que aseguran la estabilidad de la red

$$w_{ij} = w_{ji} \quad \text{y} \quad w_{ii} \geq 0$$

Obsérvese que en este caso la regla de Hebb no actúa incrementalmente, sino que de forma directa proporciona la matriz de pesos sinápticos de la red. Sin embargo, con esta regla de aprendizaje la red se satura muy rápidamente, pudiéndose almacenar hasta $0.15N$ patrones distintos (no correlacionados y con la misma probabilidad de aparición para los estados +1 y −1), siendo N el número de neuronas de la red.

Si se trata de almacenar un número mayor de patrones que el antes mencionado se presentan dos problemas: los patrones se vuelven inestables y aparecen estados espurios (estados estables que no corresponden con ningún patrón almacenado).

3.4.4.2 Regla de la Pseudoinversa

Una primera regla que mejora el rendimiento es la denominada regla de la proyección o de la pseudoinversa, propuesta para el modelo de Hopfield por L. Personnaz et al. [Personnaz 86]. Este algoritmo hace uso del cálculo de la pseudoinversa de una matriz, es válida para patrones aleatorios y no aleatorios, ortogonales y no ortogonales, asegura el almacenamiento exacto de los patrones y permite un nivel de almacenamiento de $\alpha=1$, es decir, hasta un número de patrones igual al número de neuronas de la red (no obstante si se almacenan más de $n/2$ patrones, donde n es el número de neuronas de la red, las cuencas de atracción resultan despreciables, por lo que se suele indicar como cota $\alpha=0.5$).

Sea W la matriz de pesos sinápticos de la red de Hopfield, si $\boldsymbol{x}^{\mu}$ es uno de los p vectores a memorizar, se debe cumplir

$$\text{signo}\,[W \bullet \boldsymbol{x}^{\mu}] = \boldsymbol{x}^{\mu}$$

si se trabaja con neuronas [−1,+1]. Entonces es posible sustituir la condición genérica anterior por otra particular, más restrictiva

$$W \bullet \boldsymbol{x}^{\mu} = \boldsymbol{x}^{\mu}$$

Si se colocan los **p** patrones de aprendizaje $\boldsymbol{x}^{\mu}$ como columnas de cierta matriz Σ, ésta tendrá por dimensiones $n \times p$, y la condición anterior se puede rescribir como

$$W\Sigma = \Sigma$$

El objetivo del aprendizaje será encontrar la matriz W que cumple esta condición. Si Σ fuese una matriz cuadrada de determinante no nulo, sería invertible, con lo que la solución queda

$$W = \Sigma\Sigma^{-1}$$

Pero para el caso general de una matriz no cuadrada, se demuestra que la solución es la siguiente

$$W = \Sigma\Sigma^{+} \tag{3.33}$$

siendo Σ^{+} la matriz pseudoinversa de Σ, que representa la generalización de la inversión de una matriz cuadrada para el caso de matrices rectangulares.

Para calcular la pseudoinversa puede recurrirse al método de Greville [Kohonen 89]. De este modo se dispone de una regla más potente que la de Hebb; sin embargo, el precio que hay que pagar es su complejidad de cómputo y su operación no local.

Capítulo 4

Lógica Borrosa

4.1 Introducción

Asociar la palabra “borrosa” a la palabra “lógica” podría resultar contradictorio. La lógica, en el sentido ordinario de la palabra, es una concepción de los mecanismos del pensamiento que jamás debería ser borrosa. Pero los matemáticos, profundizando en estos mecanismos del pensamiento, se dieron cuenta de que no hay, en realidad, una lógica única (por ejemplo, lógica booleana), sino tantas lógicas como se quiera, dependiendo todo de la axiomática elegida. Una vez escogida esta, todas las proposiciones que se construyan sobre ella deben cumplir rigurosamente las reglas surgidas de la misma, sin contradicción.

La mayor parte de nuestras herramientas tradicionales para el modelado formal, razonamiento y computación son de tipo *crisp*[1] o “concisas”, determinísticas y precisas en su carácter. Por *crisp* se quiere decir dicótomo, esto es, de tipo -si o no- en lugar de tipo -más o menos-. En la lógica dual convencional, por ejemplo, un enunciado puede ser cierto o falso pero nada intermedio. En teoría de conjuntos, un elemento puede o no pertenecer a un conjunto; y en optimización una solución es o no factible. La precisión asume que los parámetros de un modelo representan exactamente ya sea nuestra percepción del fenómeno modelado o las características del sistema real que ha sido modelado. Generalmente la precisión también implica modelo es inequívoco, esto es, que no contiene ambigüedades.

La certidumbre comúnmente indica que se asume que las estructuras y parámetros del modelo son definitivamente conocidos, y que no existe ninguna duda sobre sus valores u ocurrencia. Debido a que se precisa que un lenguaje de modelación sea inequívoco y no redundante, por un lado, y al mismo tiempo represente semánticamente en estos términos todo aquello que es relevante para el modelo, por otro lado, se presenta un problema. Los pensamientos y sentimientos humanos, en los cuales las ideas, imágenes y sistemas de valores están formados, antes que nada poseen más conceptos o razonamientos que palabras tiene el lenguaje que a diario utilizamos. Si se considera adicionalmente que para cierto número de conceptos se utilizan palabras similares (sinónimos) entonces se vuelve obvio que el poder (en términos teóricos) de los pensamientos y sentimientos es mucho más alto que el poder de cualquier lengua “viva” con el lenguaje lógico, entonces se encuentra que la lógica es aún más pobre.

[1] Referente a la lógica *crisp* (ver glosario).

Para los modelos o lenguajes de modelado que se basan en hechos se presentan principalmente dos problemas

1.- Las situaciones reales a menudo no son de tipo *crisp* ni determinísticas y no pueden ser descritas con precisión.

2.- La descripción completa de un sistema real a menudo requeriría datos más detallados que aquellos que un ser humano podría jamás reconocer simultáneamente, procesar y comprender.

Considérense ahora las propiedades características de los sistemas reales: Las situaciones reales son muy a menudo inciertas o vagas en cierta forma. Debido a la falta de información el estado futuro del sistema podría no ser conocido completamente. Este tipo de incertidumbre (de carácter estocástico) ha sido estudiado por la teoría probabilística y la estadística. Esta probabilidad es esencialmente frecuentista y se basa en consideraciones teóricas fijas. Es posible llamar a este tipo de incertidumbre o vaguedad *incertidumbre estocástica*, en contraste con la vaguedad concerniente a la descripción del significado semántico de los eventos, fenómenos o enunciados en sí mismos, a la que se llama *fuzzyness* [Zimmermann 90].

4.2 Teoría de Conjuntos Borrosos

La primera publicación en la teoría de conjuntos borrosos por Zadeh [Zadeh 65] mostró la clara intención por parte de dicho autor de generalizar la noción clásica de conjunto y proposición (enunciado) para acomodar la borrosidad en el sentido descrito en los párrafos anteriores.

La teoría de conjuntos borrosos provee de un marco matemático estricto en la cual fenómenos conceptualmente vagos pueden ser precisa y rigurosamente estudiados. También puede ser considerada un lenguaje de modelación bien provisto para situaciones en las cuales existen relaciones, criterios y fenómenos borrosos.

La teoría de conjuntos borrosos parte de la teoría clásica de conjuntos, añadiendo una función de pertenencia al conjunto, definida ésta como un número real comúnmente entre 0 y 1. Así, se introduce el concepto de conjunto o subconjunto borroso asociado a un determinado valor lingüístico, definido por una palabra, adjetivo o etiqueta lingüística *A*. Para cada conjunto o subconjunto borroso se define una función de pertenencia o inclusión $\mu_A(x)$, que indica el grado en que la variable *x* está incluida en el concepto representado por la etiqueta *A*. Como puede observarse en la Figura 4.1 para el valor lingüístico “Estatura de una persona” podrían definirse tres subconjuntos borrosos, cada uno identificado por una etiqueta, {Bajo, Medio, Alto}, y con una función de membresía o pertenencia $\{\mu_{Bajo}(x), \mu_{Medio}(x), \mu_{Alto}(x)\}$.

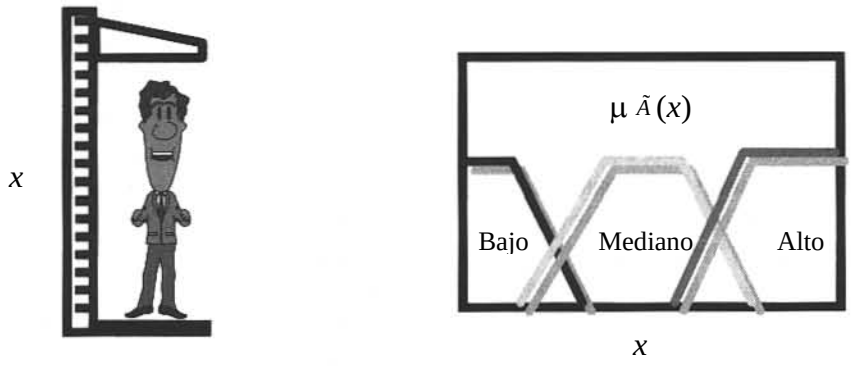

Figura 4.1: Ejemplo de conjuntos borrosos para la variable *estatura* (tomado de [Brío 02]).

4.3 Conceptos Básicos

4.3.1 Conjunto Borroso

Un conjunto clásico (conciso) normalmente se define como una colección de elementos u objetos $x \in X$ los cuales pueden ser finitos, contables o sobrecontables. Cada elemento individual puede pertenecer o no a un conjunto A, $A \subseteq X$.

Si X es una colección de objetos denotados genéricamente por x entonces un conjunto borroso $\tilde{A}$ se puede definir como un conjunto de pares ordenados

$$\tilde{A} = \{(x, \mu_{\tilde{A}}) \mid x \in X\} \tag{4.1}$$

$\mu_{\tilde{A}}(x)$ es llamada la función de membresía o grado de membresía de x en $\tilde{A}$ que mapea X al espacio de membresía M.. (cuando M. contiene únicamente los puntos, 0 y 1, $\tilde{A}$ no es borroso y es idéntico a la función característica de un conjunto no borroso). El rango de la función de membresía es un subconjunto de los números reales (desde cero hasta cierto valor máximo $< \infty$). Los elementos con un grado cero de membresía normalmente no se listan.

En la literatura es posible encontrar diferentes formas de denotar estos conjuntos:

1.- Un conjunto borroso se denota como un conjunto de pares ordenados, el primer elemento de los cuales denota el elemento en sí y el segundo el grado de membresía.

ejemplo: $\tilde{A}$ = { (1,.2),(2,.5),(3,.8),(4,1),(5,.7),(6,.3) }

2.- Un conjunto borroso se representa con sólo establecer su función de membresía (Fig. 4.2).

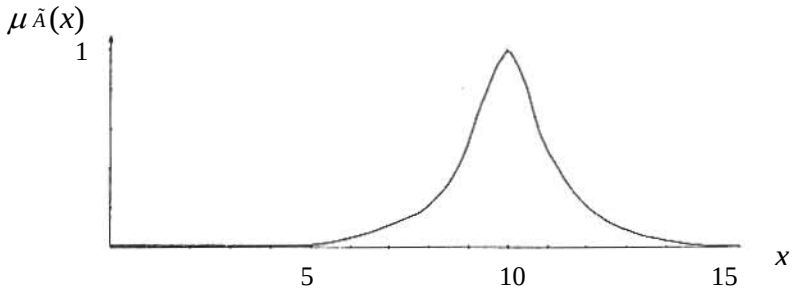

Figura 4.2: Función de membresía para los números reales cercanos a 10 (tomado de [Zimmermann 90]).

La función de membresía no se limita a valores entre 0 y 1, sin embargo, si ésta se encuentra entre dicho rango, el conjunto borroso se denomina normal. Por razones de conveniencia en adelante se hará referencia exclusivamente a conjuntos borrosos normalizados.

4.3.2 Conjunto de Grado α

El conjunto no borroso de elementos que pertenecen al conjunto borroso $\tilde{A}$ de, al menos, grado α se llama el conjunto de grado α:

$$A = \{x \in X \mid \mu_{\tilde{A}}(x) \geq \alpha\} \tag{4.2}$$

Al conjunto de grado α cuando α = 0 se le llama **conjunto soportado** de A.

4.3.3 Conjunto convexo

Se dice que un conjunto borroso es convexo si

$$\mu_{\tilde{A}}(\lambda x_1 + (1-\lambda)x_2) \geq \min(\mu_{\tilde{A}}(x_1), \mu_{\tilde{A}}(x_2)), x_1, x_2 \in X, \lambda \in [0,1] \tag{4.3}$$

4.3.4 Tipos de funciones de membresía

Para la definición de las funciones de pertenencia o membresía se utilizan convencionalmente ciertas familias de formas estándar. Las más frecuentes son la función de tipo trapezoidal, singular, triangular, S, exponencial y tipo π, que se describen a continuación:

La función de **tipo trapezoidal** se define por cuatro puntos: *a*, *b*, *c* y *d*. Esta función es cero para valores menores de *a* y mayores que *d*, vale uno entre *b* y *c*, y toma valores entre [0,1] entre *a* y *b*, y entre *c* y *d*. Se utiliza habitualmente en sistemas borrosos sencillos, pues permite definir un conjunto borroso con pocos datos y calcular su valor de pertenencia con pocos cálculos. Se emplea especialmente en sistemas basados en microprocesador, pues con similar formato pueden codificarse también funciones de tipo S, función de tipo π, triangular y singular según se distribuyan los parámetros *a*,*b*,*c* y *d* de la ecuación

$$S(x;a,b,c,d)=\begin{cases}0 & x<a\\ \left(\dfrac{x-a}{b-a}\right) & a\le x\le b\\ 1 & b\le x\le c\\ \left(\dfrac{d-x}{d-c}\right) & c\le x\le d\\ 0 & x>d\end{cases} \qquad (4.4)$$

Esta función resulta adecuada para modelar propiedades que comprenden un rango de valores para cierto parámetro o característica borrosa (p.ej. *adulto*, *normal*, *adecuado*, etc.).

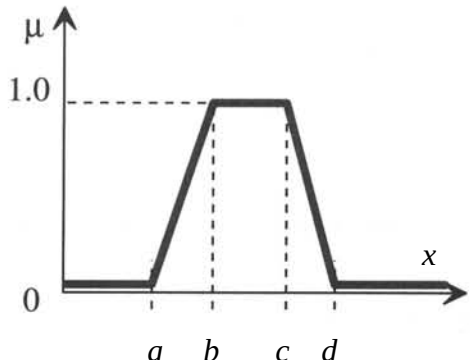

Figura 4.3: Función de membresía tipo trapezoidal (tomado de [Brío 02]).

Para modelar una **función triangular** se hace *b*=*c* en la ecuación de la función trapezoidal. Esta función es adecuada para modelar propiedades con un valor de inclusión distinto de cero para un rango de valores estrecho en torno a un punto *b*.

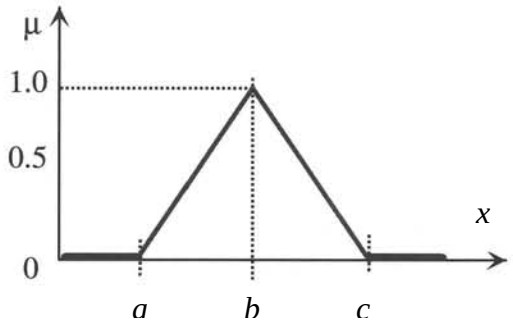

Figura 4.4: Función de membresía tipo triangular (tomado de [Brío 02]).

La función de tipo **singular** tiene valor 1 sólo para un punto *a* y 0 para el resto. Se utiliza habitualmente en sistemas borrosos simples para definir los conjuntos borrosos de

las particiones de las variables de salida, pues permite simplificar los cálculos y requiere menos memoria para almacenar la base de reglas. Se define con:

$$S(x;a)=\begin{cases}1 & x=a\\ 0 & x\neq a\end{cases} \tag{4.5}$$

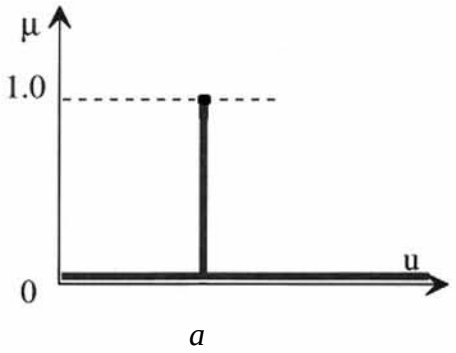

Figura 4.5: Función de membresía tipo singular (tomado de [Brío 02]).

La función de tipo S se define como:

$$S(x;a,b,c)=\begin{cases}0 & x<a\\ 2\left(\dfrac{x-a}{c-a}\right)^2 & a\le x\le b\\ 1-2\left(\dfrac{x-a}{c-a}\right)^2 & b\le x\le c\\ 1 & x>c\end{cases} \tag{4.6}$$

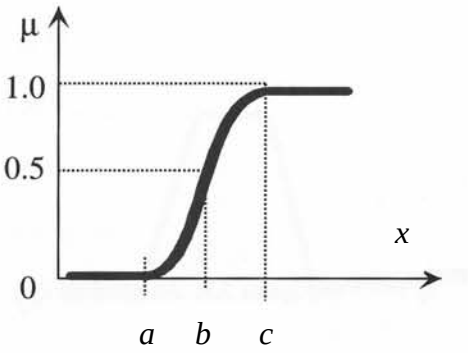

Figura 4.6: Función de membresía tipo S (tomado de [Brío 02]).

Esta función resulta adecuada para modelar propiedades como *grande*, *mucho*, *positivo*, etc. Se caracteriza por tener un valor de inclusión distinto de cero para un rango de valores pon encima de cierto punto *a*, siendo 0 por debajo de *a* y 1 para valores mayores

que *c*. Su punto de cruce (valor 0.5) es *b*=(*a*+*c*)/2; y entre los puntos *a* y *c* es de tipo cuadrático (suave). También se han utilizado funciones exponenciales para definir funciones de tipo S, como

$$S(x;k,c)=\frac{1}{1+e^{-k(x-b)}} \tag{4.7}$$

Finalmente la función tipo π puede definirse por:

$$S(x;b,c)=\begin{cases} S(x;c-b,c-b/2,c) & x\le c \\ 1-S(x;c-b,c-b/2,c) & x\ge c \end{cases} \tag{4.8}$$

tiene forma de campana y resulta adecuada para los conjuntos definidos en torno a un valor c, como *medio*, *normal*, *cero*, etc. Pueden definirse también utilizando expresiones analíticas exponenciales o cuadráticas, como la bien conocida campana de Gauss.

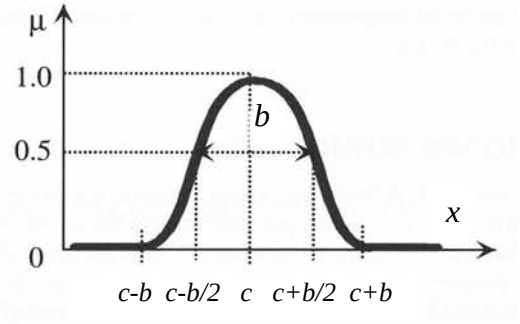

Figura 4.6: Función de membresía tipo π (tomado de [Brío 02]).

4.4 Operaciones con Conjuntos Borrosos

a) Operaciones Básicas

La función de membresía es obviamente el componente crucial de un conjunto borroso. Por lo tanto no es de sorprender que las operaciones con conjuntos borrosos se definan vía sus funciones de membresía. Inicialmente se presentan los conceptos sugeridos por Zadeh [Zadeh 65, p. 310]; sin embargo, no es la única forma de extender la teoría clásica de conjuntos de forma consistente.

4.4.1 Intersección

La función de membresía $\mu_{\tilde{C}}(x)$ de la intersección $\tilde{C}=\tilde{A}\cap\tilde{B}$ se define por

$$\mu_{\tilde{C}}(x)=\min(\mu_{\tilde{A}}(x),\mu_{\tilde{B}}(x)),x\in X \tag{4.9}$$

4.4.2 Unión

La función de membresía $\mu_{\tilde{C}}(x)$ de la unión $\tilde{C}=\tilde{A}\cup\tilde{B}$ se define por

$$\mu_{\tilde{C}}(x)=\max(\mu_{\tilde{A}}(x),\mu_{\tilde{B}}(x)),x\in X \tag{4.10}$$

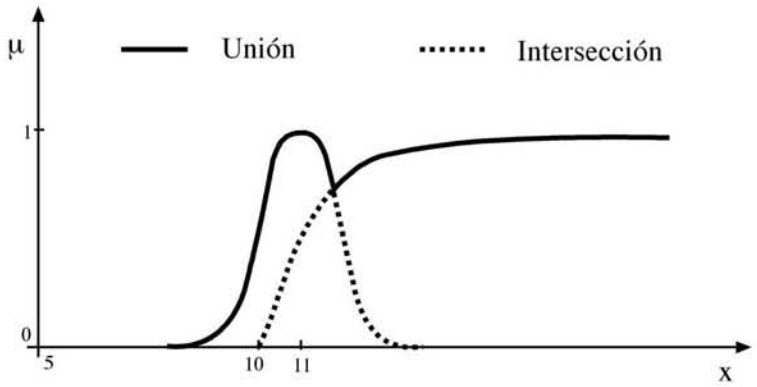

Figura 4.7: Funciones de membresía para la unión e intersección de:
a) números reales cercanos a 11 y b) Números reales mayores a 10.

4.4.2 Complemento

La función de membresía del complemento de un conjunto borroso $\tilde{A}$, $\mu_{\tilde{A}^c}(x)$ se define por

$$\mu_{\tilde{A}^c}(x) = 1 - \mu_{\tilde{A}}(x), \quad x \in X \tag{4.11}$$

b) Operadores Algebraicos

De acuerdo a [Zimmermann 90] se definen los operadores algebraicos borrosos de la siguiente forma:

4.4.3 Suma algebraica

La suma algebraica (suma probabilística) $\tilde{C} = \tilde{A} + \tilde{B}$ se define como

$$\tilde{C} = \{ (x, \mu_{\tilde{A}+\tilde{B}}(x)) \mid x \in X \}$$

donde $$\mu_{\tilde{A}+\tilde{B}}(x) = \mu_{\tilde{A}}(x) + \mu_{\tilde{B}}(x) - \mu_{\tilde{A}}(x) \cdot \mu_{\tilde{B}}(x) \tag{4.12}$$

4.4.4 Suma acotada

La suma acotada $\tilde{C} = \tilde{A} \oplus \tilde{B}$ se define como

$$\tilde{C} = \{ (x, \mu_{\tilde{A}\oplus\tilde{B}}(x)) \mid x \in X \}$$

donde $$\mu_{\tilde{A}\oplus\tilde{B}}(x) = \min(1, \mu_{\tilde{A}}(x) + \mu_{\tilde{B}}(x)) \tag{4.13}$$

4.4.5 Diferencia acotada

La suma acotada $\tilde{C} = \tilde{A} \Theta \tilde{B}$ se define como

$$\tilde{C} = \{ (x, \mu_{\tilde{A}\Theta\tilde{B}}(x)) \mid x \in X \}$$

donde $$\mu_{\tilde{A}\Theta\tilde{B}}(x) = \max(0, \mu_{\tilde{A}}(x) + \mu_{\tilde{B}}(x) - 1) \tag{4.14}$$

4.4.6 Producto algebraico

La suma acotada $\tilde{C}=\tilde{A}\cdot\tilde{B}$ se define como

$$\tilde{C}=\{\ (x,\mu_{\tilde{A}}(x)\cdot\mu_{\tilde{B}}(x))|\ x\in X\ \} \tag{4.15}$$

c) Operadores Teóricos

Es posible modelar la intersección de conjuntos borrosos, interpretada como una conjunción, como el operador *min*, y la unión, interpretada como una disyunción, con el operador *max*. Sin embargo, diversos autores han sugerido otros operadores:

4.4.7 Normas T

Los operadores mínimo, producto y suma acotada pertenecen al grupo llamado triangular o normas *t*. Los operadores de este grupo satisfacen las siguientes condiciones [Zimmermann 90]:

1. $t(0,0)=0;\ \ t(\mu_{\tilde{A}}(x),1)=t(1,\mu_{\tilde{A}}(x))=\mu_{\tilde{A}}(x)$
2. $t(\mu_{\tilde{A}}(x),\mu_{\tilde{B}}(x))\le t(\mu_{\tilde{C}}(x),\mu_{\tilde{D}}(x))$ (monotonicidad)
   $si\ \ \mu_{\tilde{A}}(x)\le\mu_{\tilde{C}}(x)\ \ y\ \ \mu_{\tilde{B}}(x)\le\mu_{\tilde{D}}(x)$
3. $t(\mu_{\tilde{A}}(x),\mu_{\tilde{B}}(x))=t(\mu_{\tilde{B}}(x),\mu_{\tilde{A}}(x))$ (conmutatividad)
4. $t(\mu_{\tilde{A}}(x),t(\mu_{\tilde{B}}(x),\mu_{\tilde{C}}(x)))=t(t(\mu_{\tilde{A}}(x),\mu_{\tilde{B}}(x)),\mu_{\tilde{C}}(x))$ (asociatividad)

Las normas *t* definen una clase general para los operadores de intersección en conjuntos borrosos. Los operadores pertenecientes a esta clase de normas *t* son, en particular, asociativos y por lo tanto es posible calcular los valores de membresía para la intersección de más de dos conjuntos borrosos aplicando recursivamente un operador de tipo norma *t*.

4.4.8 Conormas *t* o normas *s*

Para la unión de conjuntos borrosos el operador máximo, la suma algebraica y la suma acotada han sido sugeridos [Zadeh 65]. Sin embargo, al igual que para la intersección borrosa, existe una clase de operadores de agregación para la unión de conjuntos borrosos llamada conormas *t* o normas *s* que se caracterizan por cumplir las condiciones siguientes [Zimmermann 90]:

1. $s(1,1)=1;\ \ s(\mu_{\tilde{A}}(x),0)=s(0,\mu_{\tilde{A}}(x))=\mu_{\tilde{A}}(x)$
2. $s(\mu_{\tilde{A}}(x),\mu_{\tilde{B}}(x))\le s(\mu_{\tilde{C}}(x),\mu_{\tilde{D}}(x))$ (monotonicidad)
   $si\ \ \mu_{\tilde{A}}(x)\le\mu_{\tilde{C}}(x)\ \ y\ \ \mu_{\tilde{B}}(x)\le\mu_{\tilde{D}}(x)$
3. $s(\mu_{\tilde{A}}(x),\mu_{\tilde{B}}(x))=s(\mu_{\tilde{B}}(x),\mu_{\tilde{A}}(x))$ (conmutatividad)
4. $s(\mu_{\tilde{A}}(x),s(\mu_{\tilde{B}}(x),\mu_{\tilde{C}}(x)))=s(s(\mu_{\tilde{A}}(x),\mu_{\tilde{B}}(x)),\mu_{\tilde{C}}(x))$ (asociatividad)

Las normas *t* y las normas *s* se relacionan en el sentido de la dualidad lógica, de tal suerte que es posible relacionarlas de la siguiente forma [Zimmermann 90]:

$$t(\mu_{\hat{A}}(x),\mu_{\hat{B}}(x)) = 1 - s(1-\mu_{\hat{A}}(x), 1-\mu_{\hat{B}}(x)) \tag{4.16}$$

de tal suerte que cualquier norma *s* puede ser generada a partir de una norma *t* mediante esta transformación.

4.4.9 Principales normas *t* y normas *s*

De acuerdo con [Zimmermann 90, pág. 32] se presenta el siguiente listado de normas *t* y normas *s*

$t_w(\mu_{\tilde{A}}(x), \mu_{\tilde{B}}(x)) = \begin{cases} \min\{\mu_{\tilde{A}}(x), \mu_{\tilde{B}}(x)\} \text{ si } \max\{\mu_{\tilde{A}}(x), \mu_{\tilde{B}}(x)\} = 1 \\ 0 \quad \text{de lo contrario} \end{cases}$	producto drástico
$s_w(\mu_{\tilde{A}}(x), \mu_{\tilde{B}}(x)) = \begin{cases} \max\{\mu_{\tilde{A}}(x), \mu_{\tilde{B}}(x)\} \text{ si } \min\{\mu_{\tilde{A}}(x), \mu_{\tilde{B}}(x)\} = 0 \\ 1 \quad \text{de lo contrario} \end{cases}$	suma drástica
$t_1(\mu_{\tilde{A}}(x), \mu_{\tilde{B}}(x)) = \max\{0, \mu_{\tilde{A}}(x) + \mu_{\tilde{B}}(x) - 1\}$	diferencia acotada
$s_1(\mu_{\tilde{A}}(x), \mu_{\tilde{B}}(x)) = \min\{1, \mu_{\tilde{A}}(x) + \mu_{\tilde{B}}(x)\}$	suma acotada
$t_{1.5}(\mu_{\tilde{A}}(x), \mu_{\tilde{B}}(x)) = \dfrac{\mu_{\tilde{A}}(x) \cdot \mu_{\tilde{B}}(x)}{2 - [\mu_{\tilde{A}}(x) + \mu_{\tilde{B}}(x) - \mu_{\tilde{A}}(x) \cdot \mu_{\tilde{B}}(x)]}$	producto de Einstein
$s_{1.5}(\mu_{\tilde{A}}(x), \mu_{\tilde{B}}(x)) = \dfrac{\mu_{\tilde{A}}(x) + \mu_{\tilde{B}}(x)}{1 + \mu_{\tilde{A}}(x) \cdot \mu_{\tilde{B}}(x)}$	suma de Einstein
$t_2(\mu_{\tilde{A}}(x), \mu_{\tilde{B}}(x)) = \mu_{\tilde{A}}(x) \cdot \mu_{\tilde{B}}(x)$	producto algebraico
$s_2(\mu_{\tilde{A}}(x), \mu_{\tilde{B}}(x)) = \mu_{\tilde{A}}(x) + \mu_{\tilde{B}}(x) - \mu_{\tilde{A}}(x) \cdot \mu_{\tilde{B}}(x)$	suma algebraica
$t_{2.5}(\mu_{\tilde{A}}(x), \mu_{\tilde{B}}(x)) = \dfrac{\mu_{\tilde{A}}(x) \cdot \mu_{\tilde{B}}(x)}{\mu_{\tilde{A}}(x) + \mu_{\tilde{B}}(x) - \mu_{\tilde{A}}(x) \cdot \mu_{\tilde{B}}(x)}$	producto de Hamacher
$s_{2.5}(\mu_{\tilde{A}}(x), \mu_{\tilde{B}}(x)) = \dfrac{\mu_{\tilde{A}}(x) + \mu_{\tilde{B}}(x) - 2\mu_{\tilde{A}}(x) \cdot \mu_{\tilde{B}}(x)}{1 - \mu_{\tilde{A}}(x) \cdot \mu_{\tilde{B}}(x)}$	suma de Hamacher
$t_3(\mu_{\tilde{A}}(x), \mu_{\tilde{B}}(x)) = \min\{\mu_{\tilde{A}}(x), \mu_{\tilde{B}}(x)\}$	máximo
$s_3(\mu_{\tilde{A}}(x), \mu_{\tilde{B}}(x)) = \max\{\mu_{\tilde{A}}(x), \mu_{\tilde{B}}(x)\}$	mínimo

Tabla 4.1: Principales normas *t* y normas *s* (tomada de [Zimmermann 90]).

El valor calculado por los operadores anteriores se encuentra ordenado como sigue:

$$t_w \le t_1 \le t_{1.5} \le t_2 \le t_{2.5} \le t_3$$

$$s_3 \le s_{2.5} \le s_2 \le s_{1.5} \le s_1 \le s_w$$

Lo anterior implica que para cualesquiera conjuntos borrosos $\tilde{A}$ y $\tilde{B}$ en X con valores de membresía entre 0 y 1, cualquier operador de intersección que es una norma *t* está acotado por el operador *min* y el operador t_w. Una norma *s* está acotada por el operador *max* y el operador s_w respectivamente.

Los operadores *max* y *min* fijan las curvas límite para los operadores de unión e intersección y el resto de las normas *t* y normas *s* desplazan la curva respectiva hacia el 1 o 0 según sea el caso. Esto puede interpretarse como que al utilizar una norma *t* o norma *s* menos "drástica" se está modificando cierto factor de "tolerancia" de los operadores de unión e intersección borrosa (Fig. 4.8), lo cual, dependiendo de los intereses de cada aplicación en particular, puede resultar de suma utilidad.

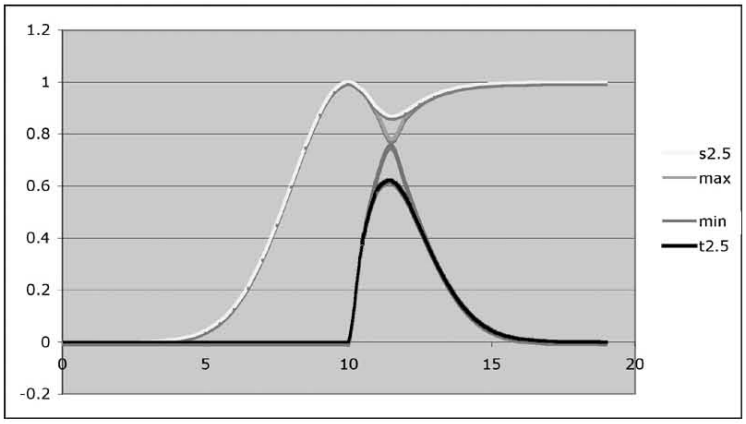

Figura 4.8: Curvas límite y variaciones de normas *t* y normas *s* correspondientes a la Figura 4.7.

4.5 Principio de Extensión

El principio de extensión permite convertir conceptos no borrosos en borrosos, siendo además la base de la inferencia en los sistemas borrosos.

Sea X el producto cartesiano de los universos de discurso $X_1 \ldots X_r$, y sean $\tilde{A}_1 \ldots \tilde{A}_r$, r conjuntos borrosos en $X_1 \ldots X_r$ respectivamente. f es una función que mapea del universo X al universo Y, con $y = f(x_1 \ldots x_r)$. Entonces el principio de extensión permite definir un conjunto borroso $\tilde{B}$ en Y por

$$\tilde{B} = \{(y, \mu_{\tilde{B}}(y)) \mid y = f(x_1, \ldots, x_r),\ (x_1, \ldots, x_r) \in X\} \qquad (4.17)$$

donde

$$\mu_{\tilde{B}}(y) = \begin{cases} \sup\limits_{(x1,\ldots,xr) \in f^{-1}(y)} \min\{\mu_{\tilde{A}_1}(x_1), \ldots, \mu_{\tilde{A}_r}(x_r)\} & si\ f^{-1}(y) \neq \varnothing \\ 0 & de\ lo\ contrario \end{cases}$$

El principio de extensión puede y ha sido modificado utilizando la suma algebraica (ecuación 4.12) en lugar del supremo, y el producto en lugar del mínimo. Sin embargo, es comúnmente utilizado como se definió anteriormente.

4.6 Relación Borrosa

Para dos universos de discurso *U* y *V*, una relación borrosa se define como un conjunto borroso *R* en el espacio $U \times V$, cuya unción de membresía se denota como $\mu_R(u,v)$ con $u \in U$ y $v \in V$.

4.7 Fuzzifier

Un *fuzzyfier* establece una relación entre puntos de entrada no borrosos al sistema $\boldsymbol{x} = (x_1,...,x_n)^T$, y sus correspondientes conjuntos borrosos *A* en *U* (las variables procedentes del exterior serán, en general, valores no borrosos, y habrá que hacerlas borrosas previamente). Se pueden utilizar diversas estrategias:

a) **Fuzzyfier singular**. El *fuzzifier* singular es el método más utilizado, principalmente en sistemas de control, y consiste en considerar los propios valores discretos como conjuntos borrosos. De otra forma, para cada valor de entada *x* se define un conjunto *A´* que lo soporta, con función de pertenencia $\mu_A(x)$, de modo que $\mu_A(x) = 1$, ($x´ = x$), y $\mu_A(x') = 0$, para todos los otros $x' \in U$ en los que $x' \neq x$.

b) **Fuzzyfier no singular**. En el *fuzzyfier* no singular se utiliza una función exponencial del tipo siguiente:

$$\mu_{A'}(x') = a \exp\left[-\left(\frac{x'-x}{\sigma}\right)^2\right] \tag{4.18}$$

Función con forma de campana, centrada en el valor *x* de entrada, de anchura σ y amplitud *a*.

4.8 Defuzzyfier

Un *defuzzyfier* es una función que transforma un conjunto borroso en *V*, con función de membresía $\mu_{B'}(y)$ resultado de un dispositivo de inferencia borrosa, en un valor no borroso $y \in V$. Para esta tarea se utilizan diversos métodos, siendo uno de los más comunes el **defuzzyfier por máximo**, definido como

$$y = \arg\sup_{y \in V}(\mu_{B'}(y)) \tag{4.19}$$

es decir, y es el punto de *V* en que $\mu_{B'}(y)$ alcanza su valor máximo, donde B´ es un único conjunto borroso que corresponde a la salida final de un dispositivo de inferencia borrosa producto de la unión de *M.* conjuntos borrosos B' y $\mu_{B'(y)}$ está definido según la ecuación

$$\mu_{B'}(y) = \mu_B{}^1(y) + \mu_B{}^2(y) + ... + \mu_B{}^M(y) \tag{4.20}$$

Capítulo 5

Creación de Mapas con Redes Neuronales

5.1 Introducción

En la actualidad el uso de redes neuronales artificiales en robots móviles se ha generalizado a casi todas las áreas, basta con mencionar trabajos como LSTM (Long Short-Term Memory [Hochreiter 99]) o las redes FENN (Fuzzy Elman Neural Network) [Zhidong 97]. LSTM es un tipo de arquitectura neuronal recurrente o realimentada, cuyas ventajas sobre otros modelos tradicionales de redes neuronales se han demostrado en áreas tales como: aprendizaje de lenguajes regulares y libres de contexto [Gers 01], predicción de series continuas de tiempo [Gers 00] o control de motores y detección de ritmo [Flake 98]. Este esquema utiliza una unidad básica de memoria llamada *memory block* o bloque de memoria (Fig. 5.1), la cual esta formada por varias unidades o neuronas en cierta disposición con realimentación. Este arreglo les permite almacenar un *estado* interno. Al agrupar varios *memory blocks*, es posible realizar tareas complejas inclusive no supervisadas, como arupamiento o *clustering* [Klapper 01].

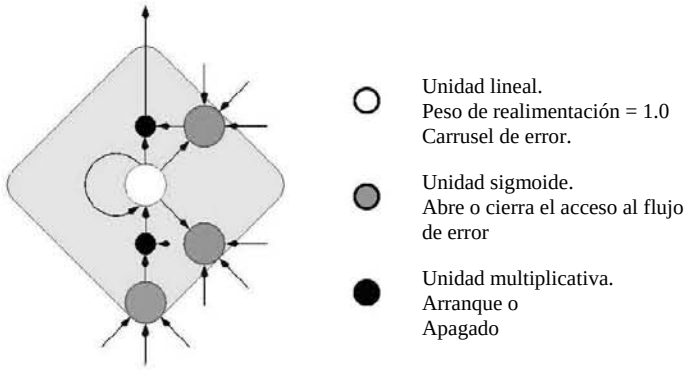

Figura 5.1: Unidad básica o *memory block* en las redes LSTM, tomada de [Hochreiter 99].

FENN es un modelo que une técnicas neuronales con técnicas borrosas. Es capaz de simular sistemas dinámicos no lineales. Consiste en una red neuronal que posee varias capas de neuronas, una de las cuales almacena reglas borrosas en forma de matrices de pesos, otra capa se encarga de evaluar las reglas borrosas y otra más implementa un *defuzzyfier* que realimenta a la entrada el resultado no borroso (Fig. 5.2).

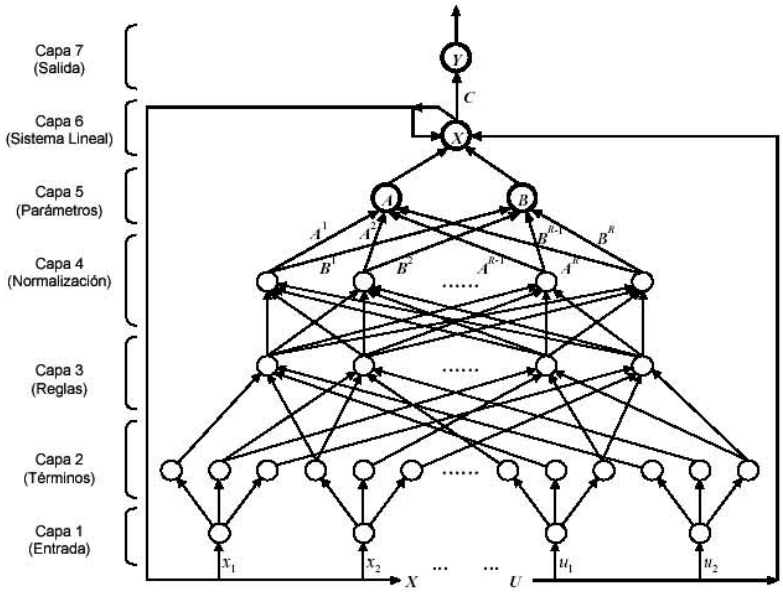

Figura 5.2: Estructura de una red FENN, tomada de [Zhidong 97].

Como puede observarse la fusión de técnicas borrosas con técnicas neuronales y cierto grado de realimentación ha mejorado el desempeño de los modelos tradicionales.

En cuanto a la aplicación de redes neuronales en la creación de mapas topológicos, ésta puede hacerse en tres niveles:

1) **Movimiento autónomo del robot**. Es posible utilizar técnicas de redes neuronales para guiar a un robot móvil con el objetivo de que se mueva libremente en el ambiente de operación y pueda ser capaz, por sí mismo, de navegar sin dañarse ni dañar el entorno, así como de tomar suficientes lecturas para poder elaborar algún tipo de mapa.

2) **Creación del mapa**. También es posible utilizar técnicas de redes neuronales para procesar las muestras obtenidas por el robot y elaborar un mapa del entorno.

3) **Localización**. Finalmente una vez que ya se ha elaborado el mapa del ambiente es factible utilizar nuevamente técnicas de redes neuronales para tratar de encontrar la ubicación real del robot en función de las observaciones que realice sobre su actual posición.

En el presente trabajo se aborda una metodología para implementar el movimiento autónomo del robot y la creación de los mapas métrico y topológico utilizando redes neuronales, alimentadas en ambos casos exclusivamente con mediciones realizadas con los sonares del robot y sin ningún tipo de realimentación.

En la parte de localización se hace una breve comparación de dos técnicas, la primera utilizando redes neuronales autoasociativas (que trataron de memorizar vistas locales), y la segunda mediante localización basada en procesamiento digital de imágenes (en todos los casos se tomó como única entrada las mediciones realizadas con los sonares del robot).

No obstante el tema de la localización no constituye el motivo central del presente trabajo, se realizó lo anterior para evaluar las capacidades de las técnicas neuronales artificiales para el problema SLAM en todos sus aspectos.

5.2 Metodología

Debido a que en el presente trabajo se intentó evaluar la factibilidad y eficiencia de las redes neuronales artificiales para realizar tareas de movimiento autónomo, creación de mapas y auto localización, se hizo imprescindible contar con un marco de referencia que permitiese comparar procedimientos que hicieran uso de dichas técnicas contra los métodos "tradicionales" que no las utilizan.

Inicialmente se implementaron en un ambiente controlado (simulador Roc2) algoritmos de movimiento autónomo y creación de mapas que no involucraban redes neuronales artificiales. Lo anterior resultó indispensable, ya que en todo ambiente real de operación existe una gran cantidad de factores que afectan tanto a los movimientos del robot como a las mediciones de sus sensores. Si se desea implementar un método que haga uso de algún algoritmo, por ejemplo un seguidor de paredes, directamente en dicho ambiente real de operación, resultaría casi imposible tomar en cuenta todos los posibles factores que intervendrían en el proceso. Por el contrario, en un ambiente "idealizado" resulta relativamente simple implementar algoritmos que realicen eficientemente las tareas deseadas y cuando se desee, es posible agregar errores a los movimientos o mediciones del robot. De igual forma se utilizó un ambiente controlado para obtener mediciones con los sonares del robot "virtual" y tratar de asociar, mediante algún tipo de red neuronal, su posición y orientación a las observaciones hechas por el robot, con la finalidad de evaluar directamente en este ambiente idealizado la factibilidad del uso de tales herramientas en la auto localización.

Posteriormente se probaron las mismas rutinas agregando cierto grado de error en el ambiente controlado, y finalmente en el ambiente real de trabajo y con el robot real. Se determinaron los tipos de errores cometidos y los factores externos que alteraban la correcta operación del robot. Con base en tales errores se implementó una solución utilizando redes neuronales artificiales y se probó nuevamente en el ambiente real, determinando la eficiencia de los nuevos métodos.

De acuerdo con lo anterior la metodología seguida fue la siguiente:

1. Se implementó dentro de un ambiente simulado una rutina de movimiento autónomo (seguimiento de paredes) y otra para la creación del mapa para el robot virtual, ambas sin redes neuronales y se probó su desempeño.

2. Una vez que se hubieron probado las rutinas de movimiento autónomo del robot y creación del mapa, se hizo que un robot virtual fuera tomando lecturas del entorno mientras se desplazaba dentro del mismo.

3. Con base en las lecturas realizadas se diseñó e implementó una red neuronal en el programa *Matlab* (www.mathworks.com) (que tomaba como entrada las lecturas

realizadas por los sonares en cierta ubicación, y proporcionaba como salidas el número de nodo del mapa correspondiente a dicha ubicación y la orientación con respecto al marco de referencia utilizado.

4. Se analizaron los resultados y se probó nuevamente la red agregando ruido, tanto a los movimientos del robot como a las mediciones de los sensores del mismo y se analizaron nuevamente los resultados obtenidos.

5. Se probaron los algoritmos de movimiento autónomo y creación del mapa, desarrollados en el ambiente virtual, ahora con el robot real y en el ambiente real de operación. Se analizaron los resultados obtenidos.

6. Se desarrollaron métodos de movimiento autónomo y creación del mapa utilizando redes neuronales artificiales y se probó su eficiencia en el ambiente real de operación.

7. Se compararon los resultados obtenidos sin redes neuronales contra los obtenidos haciendo uso de ellas.

8. Con las mediciones obtenidas en el ambiente real se probó la red neuronal que indicaba la posición y orientación del robot. Se analizaron los resultados.

9. Se utilizó un método de localización sin redes neuronales y se probó en el ambiente real.

10. Se compararon los resultados globales y se determinó en cuáles casos resultó mejor el desempeño utilizando una red neuronal.

11. Se listaron las conclusiones del trabajo y se sugirieron posibles mejoras al método.

5.3 Caso de Estudio

En el Laboratorio de Biorrobótica de la Facultad de Ingeniería de la UNAM se llevan a cabo el desarrollo y prueba de robots móviles en diferentes categorías: robots seguidores de línea, velocidad, laberinto, lego mindstorms y robocup entre otras.

Con la adquisición de un robot de la marca BeeSoft modelo B14 bautizado como “T×8“ surgió la inquietud de desarrollar un proyecto que hiciera uso de este nuevo robot y se acordó diseñar y poner en operación un robot guía para los visitantes del museo de las ciencias *Universum* (ubicado en el circuito cultural, s.n., Ciudad Universitaria, México, D.F.) cuyo objetivo sería desplazarse a lo largo de una o varias salas, presentar información a los asistentes e interaccionar con los mismos de acuerdo con sus capacidades.

De acuerdo con el objetivo planteado anteriormente, puede verse que definitivamente resultaba indispensable dotar al T×8 de capacidades de movimiento autónomo, planeación de rutas, evasión de obstáculos, auto localización, representaciones internas de conceptos y reglas, capacidades de interacción hombre-robot, etc., es decir, se

requería implementar una arquitectura de operación, que por familiaridad se eligió fuese la de ViRbot ([Savage 98], ver capítulo 2).

Por otro lado se tenía el problema de que, dada la continua afluencia de visitantes al museo de las ciencias, de entrada no sería factible realizar el desarrollo y las pruebas iniciales del sistema dentro del propio museo. Por ello se eligió el mismo Laboratorio de Biorrobótica para hacer el desarrollo y prueba de los sistemas del robot.

Figura 5.3: Laboratorio de Biorrobótica de la D.E.P.F.I. y robot T×8.

5.4 El Robot T×8

El robot T×8 es un robot móvil modelo B14 de la marca BeeSoft. Cuenta con un sistema motriz (localizado en la base del mismo) que utiliza dos motores de movimiento continuo para realizar 2 clases de movimientos: rotación de las tres ruedas sobre su propio eje y giro de las mismas para avanzar o retroceder. Este sistema de locomoción (también conocido como sistema de manejo síncrono o synchro drive [Dudek 00, pp. 22]) le permite al T×8 girar sobre su propio eje y, dado que se puede controlar independientemente los movimientos de cada motor, es capaz realizar movimientos complejos que involucren translación y rotación simultánea, con lo que puede avanzar trazando curvas o ajustándose a ellas.

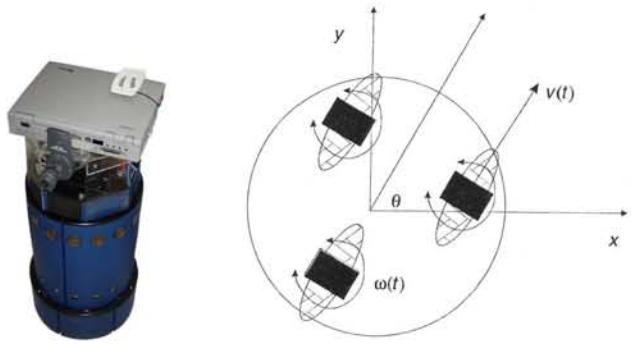

Figura 5.4: Robot T×8 y su sistema de locomoción (tomado de [Dudek 00, pp. 23]).

En su estructura media cuenta con un anillo de 16 sonares equidistantes y un anillo de 16 sensores infrarrojos también equidistantes. El anillo de sonares se encuentra situado a una altura de 80 cm. del piso mientras que el de infrarrojos se encuentra situado a una altura de 30 cm. del mismo. La dirección tanto de los sonares como de los infrarrojos no apunta directamente hacia el frente (como se aprecia en la Figura 5.5), es decir, no obstante cada uno de ellos se encuentra situado a 22.5^o (igual a 360^o /16 sonares) con respecto del siguiente sonar en el anillo, el primero no observa en línea recta hacia los 0^o (frente del robot), sino que se encuentra desviado a 11.25^o, mientras que el último sonar (el 16) mira hacia 348.75^o (-11.25^o). Lo anterior resulta relevante ya que el robot no "ve" tanto con sus sonares como con sus sensores infrarrojos directamente hacia el frente, atrás o a sus costados. Esto deberá ser tomado en cuenta al momento de implementar el algoritmo de seguimiento de paredes.

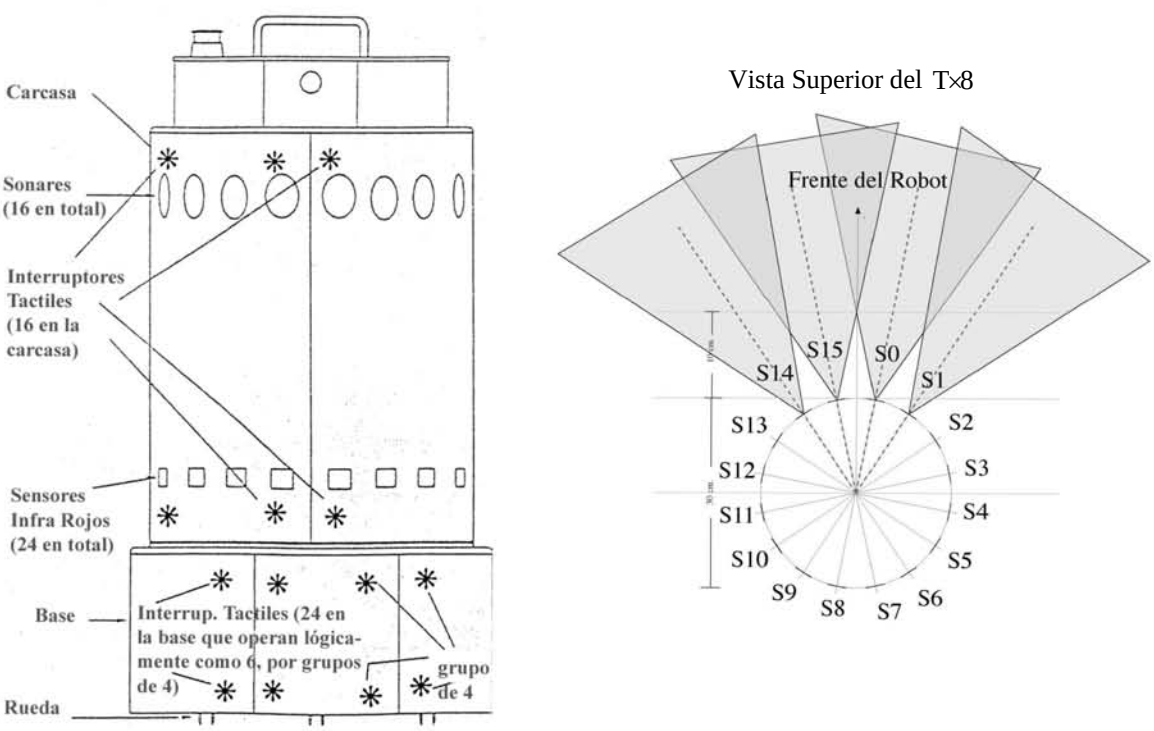

Figura 5.5: Sensores del T×8 y ubicación de sus sonares.

El robot posee en su interior una tarjeta madre con un procesador Intel 386, un disco duro de 10Gb, tarjeta de red a 100 mbps, una tarjeta de comunicaciones seriales y una tarjeta de video VGA para poder conectar un monitor externo. Posee instalado el sistema operativo linux RedHat 1.0 y un sistema de control que permite recibir comandos remotamente, utilizando para ello una computadora portátil (ubicada en la parte superior del robot) conectada en red directamente con el robot y al mismo tiempo con la red del laboratorio a través de una tarjeta de red inalámbrica (cuya antena se aprecia sobre la laptop en la Figura 5.4).

El sistema de control recibe comandos del exterior en el mismo formato que se utiliza dentro del simulador Roc2, lo que facilita las pruebas en el robot real de las rutinas desarrolladas en el simulador.

5.5 El Ambiente de Trabajo

En la Figura 5.3 se muestra el laboratorio. Contiene mobiliario de una gran diversidad de materiales, incluyendo madera, metal, plástico, vidrio y objetos de cartón.

Inicialmente para poder con una representación realista del ambiente de trabajo del T×8 se realizó una modelación del ambiente de trabajo en el programa Corel Bryce 5.0 que se muestra a continuación:

Figura 5.6: Modelado en 3D del Laboratorio de Biorrobótica.

Con base en la información utilizada para generar el modelo se obtuvo una vista en planta del mismo que sirvió para generar el archivo de objetos que se alimentaría al simulador en una etapa posterior al realizar la simulación.

Cabe aclarar que el ambiente simulado se redujo exclusivamente al espacio de navegación del robot, ya que, no obstante al fondo del laboratorio se encuentra un área dedicada para hacer investigación con robots que juegan futbol, la separa un pasillo estrecho por donde no se desea que cruce el robot T×8.

5.6 El Simulador Roc2

El Robot Command Center 2.0 (Roc2) es una herramienta de simulación desarrollada dentro del Laboratorio de Biorrobótica de la F.I. que permite probar los algoritmos diseñados en robots virtuales antes de implantarlos en los robots reales.

Dentro de las capacidades con que cuenta el Roc2 se encuentra la posibilidad de leer representaciones de diferentes ambientes de trabajo en forma de polígonos tridimensionales (mediante la importación de un archivo con cierto formato), tomar lecturas con los sensores del robot virtual acerca de este ambiente, tomar y soltar objetos con un brazo mecánico simulado, realizar desplazamientos dentro del ambiente e interactuar con otros agentes o robots virtuales que pueden ser añadidos directamente por el usuario durante la simulación, así como añadir objetos no contemplados en el modelo original para simular situaciones no previstas.

La gran ventaja de esta herramienta es que posee un módulo de desarrollo de robots virtuales donde se especifican la forma, dimensiones, número, tipo y ubicación de los sensores del robot, color, etc., y por defecto utiliza una representación virtual del T×8.

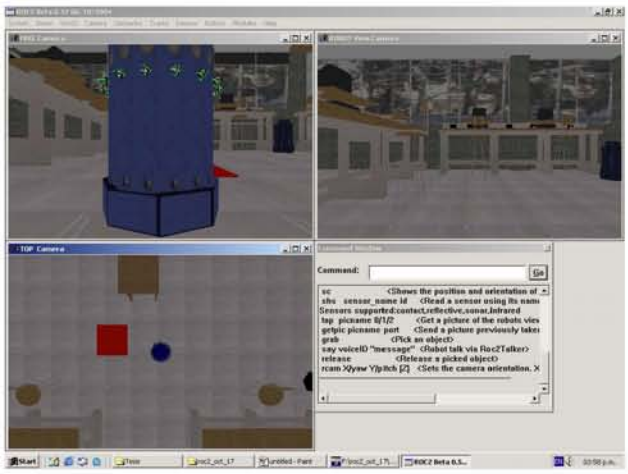

Figura 5.7: Interfaz de usuario del Roc2 y T×8 *virtual.*

Para poder incorporar rutinas de usuario el Roc2 utiliza una librería dinámica (llamada Roc2user.DLL) que se lee en tiempo de ejecución. Para agregar una rutina de usuario basta con definirla dentro del archivo Roc2User.cpp y generar nuevamente el DLL. Para invocar la nueva función basta con teclear: *user* <no. función> "<Param1> ..<Param n>" en la ventana del Roc2 donde se introducen los comandos para el robot.

Dentro de los comandos básicos que incorpora el Roc2 para el control de los robots virtuales destacan por su importancia tres de ellos (Figura 5.8):

```
//user 5 "distance angle time"
//example: user 5 "30 -0.7071 2"
void user5(char *string){
       float sonar;
       float coo[4],x,y,theta;

       //it gets the robot's position
       show_coordinates("Tx8",coo);
       x=coo[0]; y=coo[1];theta=coo[2];

       printf("Robot's Position x %.3f y %.3f theta
       %.3f\n",x,y,theta);

       getusrpos("Tx8",coo);

       printf("User Position x=%.3f y=%.3f\n",coo[0],coo[1]);

       // it moves the robot the asked distance and angle
       move_robot("Tx8",8,2,1);

       // it reads sonar 1
       sonar=show_sensor("Tx8", "sonar", 1);

       printf("Sonar 1 %f\n",sonar);
}
```

Figura 5.8: Ejemplo de función de usuario.

- *show_coordinates* (coordenadas): devuelve en un arreglo de números de punto flotante la posición estima del robot de acuerdo con su odometría interna.
- *move_robot* (distancia, ángulo, velocidad): mueve el robot virtual una distancia, ángulo y con la velocidad indicada en los parámetros.
- *show_sensor* (tipo_de_sensor, no_de_sensor): devuelve la medición realizada por el número y tipo de sensor especificado.

Debido a que en el simulador pueden coexistir varios robots, es necesario indicar en cada comando el identificador del robot al cual se dirige la acción solicitada.

Con estos tres sencillos comandos es posible implementar una infinidad de rutinas de movimiento y reacción. Cabe recordar que el robot real T×8 recibe exactamente estos comandos con el mismo formato lo cual facilita enormemente el proceso de prueba.

5.7 Creación del Mapa Métrico en el Ambiente Virtual

5.7.1 Representación del Ambiente Virtual

En primera instancia el robot virtual debe crear el mapa de entorno que lo rodea.

Para poder lograr lo anterior primero se debe introducir al simulador el modelo del ambiente de trabajo del robot. Esto se hace mediante un archivo donde se definen las características y dimensiones del Laboratorio de Biorrobótica (en dos dimensiones):

```
( limit_area lbr 0.00 0.00 0.00 76.9 53.1 76.9 53.1 0.00 )
( dimensions lbr 53.1 76.9 )
( polygon wall lbr pared1 0.00 0.00 0.00 0.30 53.1 0.30 53.1 0.00 )
( polygon wall lbr pared2 0.00 0.00 0.00 76.9 0.30 76.9 0.30 0.00 )
( polygon wall lbr pared3 0.00 76.6 0.00 76.9 53.1 76.9 53.1 76.6 )
( polygon wall lbr pared4 52.8 0.00 52.8 76.9 53.1 76.9 53.1 0.00 )
( polygon object lbr locker 0.3 0.3 0.3 6.6 9.9 6.6 9.9 0.3 )
( polygon object lbr mesas1 0.3 16.5 0.3 31.6 12.0 31.6 12.0 16.5 )
( polygon object lbr mesas2 0.3 41.6 0.3 55.1 12.1 55.1 12.1 41.6 )
( polygon object lbr mesas3 0.3 68.1 0.3 76.3 27.0 76.3 27.0 68.1 )
( polygon object lbr mesa4  32.7 68.0 32.7 76.3 52.5 76.3 52.5 68.1 )
( polygon object lbr mesa5  44.4 36.6 44.4 54.6 52.5 54.6 52.5 36.6 )
( polygon object lbr mesa6  36.9 20.7 36.9 28.2 52.5 28.2 52.5 20.7 )
( polygon object lbr libros 47.5 11.2 47.5 20.7 52.5 20.7 52.5 11.2 )
( polygon object lbr mesa7  24.7 0.3 24.7 9.2 36.7 9.2 36.7 0.3 )
( polygon object lbr mesa8  9.9 0.3 9.9 7.4 24.7 7.4 24.7 0.3 )
( polygon object lbr caja  22.0 28.3 22.0 33.4 27.1 33.4 27.1 28.3 )
( polygon object lbr mesac 22.0 33.4 22.0 61.9 31.7 61.9 31.7 33.4 )
( polygon object lbr bote  27.1 30.3 27.1 33.4 29.2 33.4 29.2 30.3 )
```

Figura 5.9: Archivo con definiciones del Laboratorio *virtual* de Biorrobótica.

Una vez realizado el archivo anterior, se procedió a alimentarlo al simulador para poder comenzar el proceso de desarrollo de las rutinas de movimiento autónomo en el ambiente virtual.

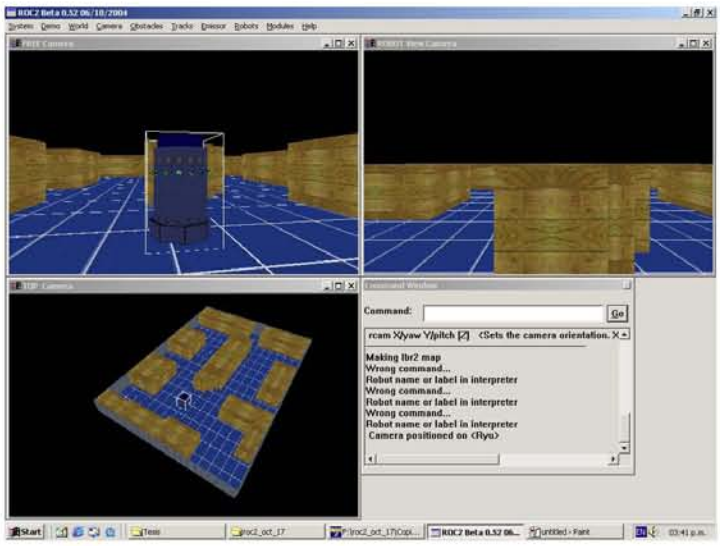

Figura 5.10: Simulador con el modelo virtual del ambiente de trabajo y T×8 virtual.

5.7.2 Movimiento Autónomo

Como rutina de movimiento autónomo se utilizó un algoritmo simple de seguimiento de paredes usando la mano izquierda o derecha) ampliamente utilizado en la resolución de laberintos).

1. Avance en línea recta hasta que encuentre una pared al frente
2. Gire 90^o a la derecha
3. Si hay espacio a su izquierda
    3.1. Gire 90^o en esa dirección y avance un paso
   de lo contrario
    3.2. Si hay espacio al frente
        3.2.1. Avance un paso
    de lo contrario
        3.2.2. Si hay espacio a su derecha
            3.2.2.1. Gire 90^o en esa dirección y avance un paso
        de lo contrario
            3.2.2.2 Gire 180^o y avance un paso

4. Repita desde 3. hasta que regrese a la posición del punto 2.

Figura 5.11: Algoritmo de seguimiento de paredes usando la mano izquierda.

5.7.3 Implementación del Algoritmo

Para poder implementar el algoritmo anterior en el robot, se debe tener en cuenta que no es capaz de "ver" directamente hacia el frente, los costados ni hacia atrás con sus sonares, ya que los dos más cercanos a dichas orientaciones se encuentran desviados a $+11.25^o$ y -11.25^o respectivamente. Por tal motivo se tuvieron que combinar las mediciones para determinar la distancia libre en cada dirección y evitar con ello que el robot pudiese colisionar con los objetos del ambiente.

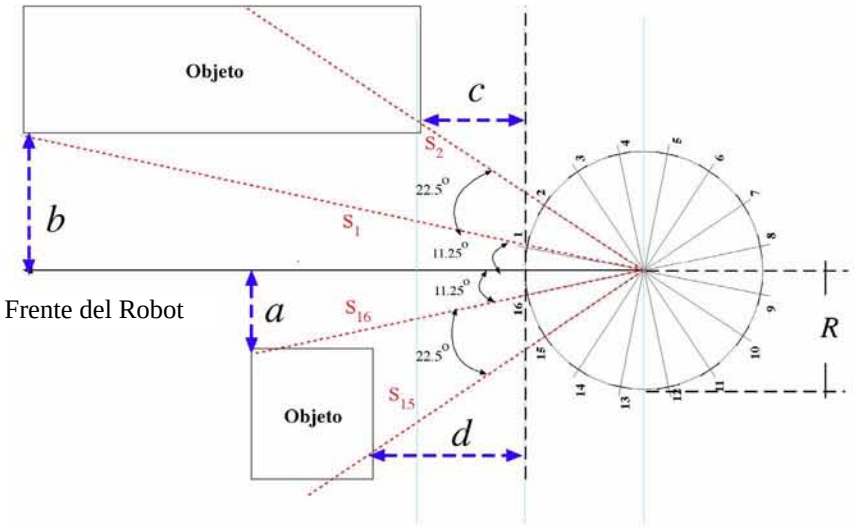

Figura 5.12: Cálculo de distancia navegable para el robot.

De acuerdo con la Figura 5.12 es posible hacer que el robot avance una distancia *D* hacia el frente del mismo si se cumplen las siguientes condiciones:

$a > (R+\alpha)$ (5.1), $b > (R+\alpha)$ (5.2), $c > (D+\alpha)$ (5.3) y $d > (D+\alpha)$ (5.4),

donde α es un margen de seguridad , *R* es el radio del robot, y *a*, *b*, *c* y *d* se calculan con:

$a = S_{16} \text{ sen } (11.25^{o})$ (5.5), $b = S_1 \text{ sen } (11.25^{o})$ (5.6)

$c = S_{15} \cos (33.75^{o}) - R$ (5.7), $d = s_2 \text{ Sen } (33.75^{o}) - R$ (5.8)

mientras que S_1, S_2, S_{15} y S_{16} corresponden con las lecturas de los sonares que apuntan hacia el frente del robot (las ecuaciones y condiciones anteriores se aplican de igual forma para detectar la distancia libre del robot hacia las direcciones restantes: izquierda, derecha y atrás, remplazando los índices de los sonares por aquellos que se dirigen en la dirección de interés).

Las condiciones (5.1) y (5.2) garantizan que el robot puede pasar por el espacio libre delante de él, mientras que las condiciones (5.3) y (5.4) pretenden evitar que existan objetos muy próximos al robot que queden fuera del alcance de los sonares más próximos a la dirección del movimiento y que le impidan desplazarse en la dirección prevista.

Finalmente para acelerar el recorrido por el ambiente se definió una distancia o "paso" de avance del robot igual al radio del mismo (15 cm.), lo anterior evita tener que recalcular a cada instante la distancia libre al frente del robot.

Tanto en el robot virtual como en el real, la rutina que implementa el movimiento del robot calcula un movimiento uniformemente acelerado del mismo (hasta la mitad del recorrido) y luego un movimiento uniformemente desacelerado. Debido a lo anterior, si se elige un paso muy pequeño, el robot invertirá demasiado tiempo entre movimientos sucesivos.

Cuando la longitud del paso es mayor al radio del robot (más la distancia de seguridad), si el robot se encuentra prácticamente en la esquina de una pared u objeto y no alcanza aún a dar la vuelta por una fracción muy pequeña, el siguiente paso hacia el frente lo puede dejar demasiado alejado de la esquina (si el robot avanza sólo un poco más de la cuenta debido a errores en el desplazamiento), de tal suerte que al girar para seguir el muro u objeto, se le indique nuevamente que tiene que girar porque sigue existiendo espacio libre a su izquierda (y más si los objetos no se hayan alineados con el robot) lo cual en ocasiones lo puede hacer girar indefinidamente. Para evitar esto basta con agregar al algoritmo una nueva condición que impida que el robot gire dos veces sucesivas en la misma dirección.

1. Avance en línea recta hasta que encuentre una pared al frente
2. Gire 90^o a la derecha
3. Si hay espacio a su izquierda ***y no acaba de girar en esta dirección***
    3.1. Gire 90^o en esa dirección y avance un paso
    de lo contrario
    3.2. Si hay espacio al frente
        3.2.1. Avance un paso
        de lo contrario
        3.2.2. Si hay espacio a su derecha ***y no acaba de girar en esa dirección***
            3.2.2.1. Gire 90^o en esa dirección y avance un paso
            de lo contrario
            3.2.2.2 Gire 180^o y avance un paso

4. Repita desde 3. hasta que regrese a la posición del punto 2.

Figura 5.13: Algoritmo de seguimiento de paredes que previene ciclos.

5.7.4 Prueba del Algoritmo en el Ambiente Virtual

Se probó el algoritmo de movimiento en el ambiente virtual y, con las adecuaciones anteriores para prevenir ciclos, funcionó sin mayores problemas. A continuación se muestra el recorrido efectuado por el robot virtual utilizando dicho algoritmo.

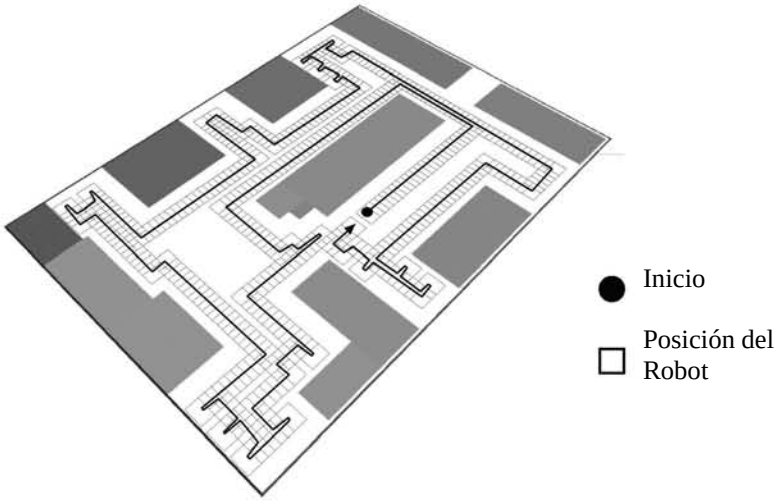

Figura 5.14: Prueba del movimiento autónomo del robot en el ambiente virtual.

5.7.5 Creación del Mapa en el Ambiente Virtual

Para poder utilizar el simulador fue necesario crear un archivo con la representación métrica de los objetos del ambiente de trabajo del robot virtual y definir sus dimensiones, por lo que de antemano se conocen las dimensiones del espacio de navegación.

Para simplificar el proceso de creación del mapa métrico y topológico se utilizó una representación en forma de malla rectangular (mapa sensorial, ver capítulo 2, sección 2.7.1) en forma de celdas de tamaño fijo (por omisión del diámetro del robot).

Para representar el mapa topológico basta con asignar a cada posición o celda dentro del mapa un valor lógico que indique si son navegables o no, es decir, si existe algún objeto dentro de esa región que impida al robot desplazarse libremente dentro del espacio comprendido por la celda. De esta forma es posible representar las celdas no navegables como objetos rectangulares de dimensiones iguales a la celda en cuestión y localizadas en el centroide de cada celda dentro del mapa.

Para determinar qué celdas se encuentran ocupadas se aprovechó que el robot debe realizar lecturas con cuatro sonares para cada dirección (16 sensores en total) al desplazarse de forma autónoma por el ambiente. Si se supone que el mapa inicialmente se encuentra vacío y por cada lectura de sonar (que necesariamente implica la detección de algún objeto en su trayectoria) se determina el punto de coordenadas (x,y) que corresponde con la lectura obtenida, es posible asumir que existe algún objeto dentro de la celda que corresponde con el punto (x,y) encontrado (Figura 5.15).

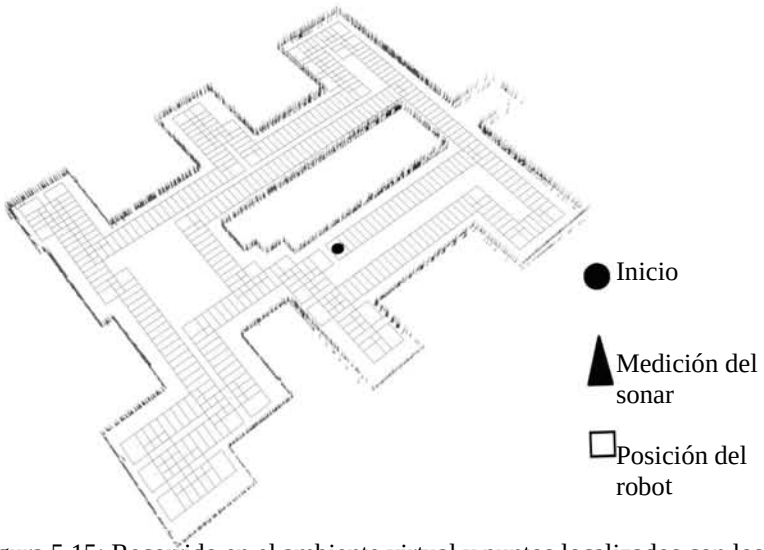

Figura 5.15: Recorrido en el ambiente virtual y puntos localizados con los Sonares.

Es posible reutilizar la información obtenida por los sonares en la navegación autónoma, para determinar las celdas que se hallan libres de obstáculos y encontrar el mapa métrico y topológico. Se indicó al robot previamente a la ejecución del algoritmo su posición y orientación dentro del ambiente virtual. Si bien esto no es necesario, se hizo para poder generar un mapa con la orientación correcta y compararlo contra un mapa "ideal".

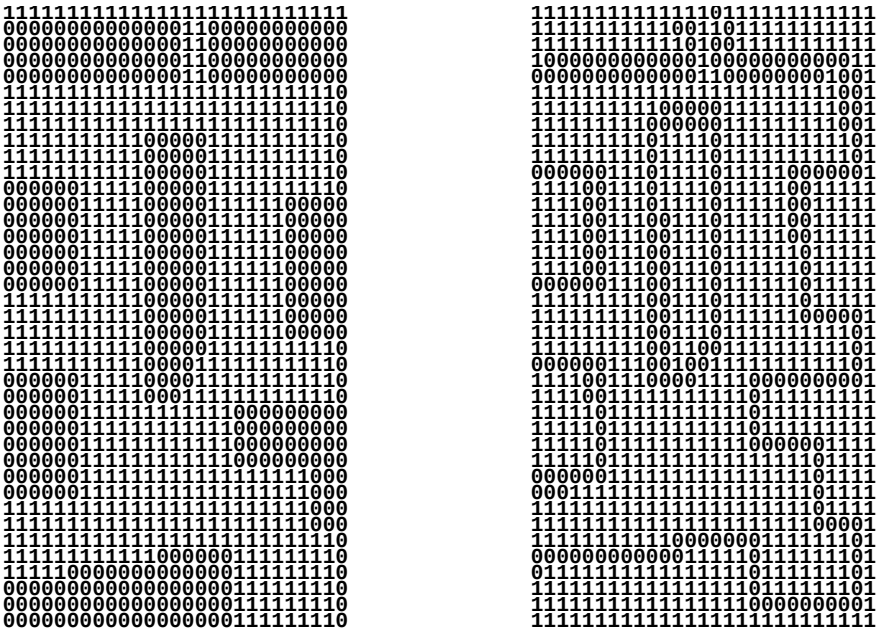

Figura 5.16: Mapa ideal (izquierda) generado a partir del archivo con objetos del mundo virtual y mapa elaborado con el algoritmo de movimiento autónomo (derecha).

Como se puede observar en la figura anterior, este método deja muchas celdas libres al interior de los obstáculos, ya que el sonar únicamente indicará una medición para el contorno de los mismos. Lo anterior no representa un problema si el robot recorre prácticamente todo el espacio y no alcanza a "cerrar" o marcar como no navegables todas las celdas que corresponden a la periferia de los objetos. Si se elige un paso de avance demasiado grande (digamos cuatro veces el radio del robot), puede haber celdas en el perímetro de los objetos que no sean alcanzadas por ninguna medición de los sonares (recuérdese que estos se hallan dispuestos sólo en ciertos ángulos con respecto al frente del robot). De ser este el caso se presentaría el problema de que se marcarían como navegables aquellas celdas al interior de los objetos y si alguna quedara libre en la periferia del mismo, además de hacer que el modelo final posea mas nodos navegables de los necesarios, sería teóricamente posible pedirle al robot que se moviera al interior de los objetos grandes (digamos una caja o mesa) y él encontraría una ruta factible e intentaría realizar su tarea.

Otro criterio que se puede utilizar consiste en marcar inicialmente todas las celdas como ocupadas y "limpiarlas" conforme el robot se mueve en el ambiente. Esto puede hacerse trazando una línea desde el centro del robot y hasta el punto indicado por la medición de cada sonar. Es posible marcar como libres aquellas celdas por las cuales cruza la línea o usar alguno de los métodos utilizados en graficación computacional para trazar líneas, como base para calcular cuáles celdas son susceptibles de ser "limpiadas". Este método es similar a un proceso de "trasquilado" o "rasurado" puesto que cada medición marca cierto número de celdas no navegables como navegables.

En este caso se utilizó el método especificado en [González 02, pp. 87] para trazar líneas. Se traza una línea horizontal a ambos lados del centroide de cada nodo (de longitud igual a un cuarto del ancho de la celda) y se determina dónde cruza la línea que se pretende dibujar a estas pequeñas líneas horizontales. Aquellas celdas donde se cruzan ambas líneas se marcan como libres, produciendo el mapa que se muestra a continuación:

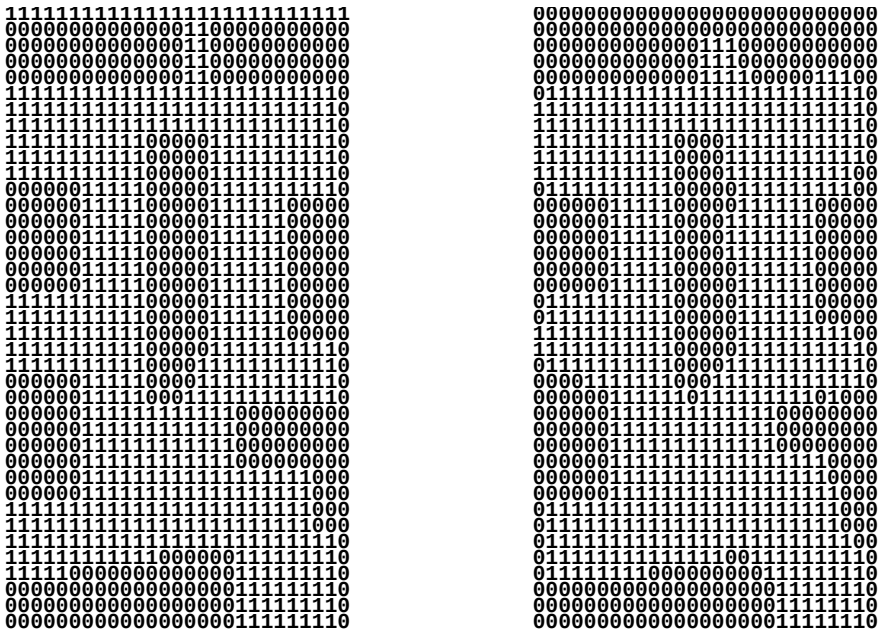

Figura 5.17: Mapa ideal (izquierda) y mapa elaborado con el algoritmo de movimiento autónomo con “rasurado” (derecha).

Como puede observarse en la figura anterior, la única limitante es que en algunas ocasiones las esquinas de los objetos rectangulares pueden aparecer recortadas, lo anterior se debe a que la línea de visión del sonar puede pasar oblicuamente muy cerca de una esquina pero sin tocarla y al momento de limpiar las celdas se eligió la celda de la esquina del objeto para ser “rasurada”.

5.8 Auto Localización en el Ambiente Virtual

Una vez resuelto el problema de la navegación autónoma y la creación del mapa se dirigieron los esfuerzos a diseñar un método de auto localización del robot virtual en este ambiente controlado, para ello se pretende utilizar tanto las observaciones realizadas por los sonares del robot en cada posición de su recorrido como las posiciones y orientaciones asociadas con cada ubicación, para entrenar una red neuronal que proporcione la ubicación (o nodo) y la orientación del robot en función exclusivamente de las observaciones realizadas con sus sonares.

5.8.1 Determinación de la Arquitectura de la Red

Se propuso una red tipo progresiva de tres capas: una capa de entrada donde se alimenten las observaciones obtenidas por los 16 sonares, una capa intermedia con un número de unidades por determinar, y una capa de salida con dos unidades: la primera debe proporcionar un número de nodo visitado o índice, asociado con la posición en la cual se encuentra el robot (no su posición espacial *x*,*y*), y la segunda la orientación del robot con respecto al mapa.

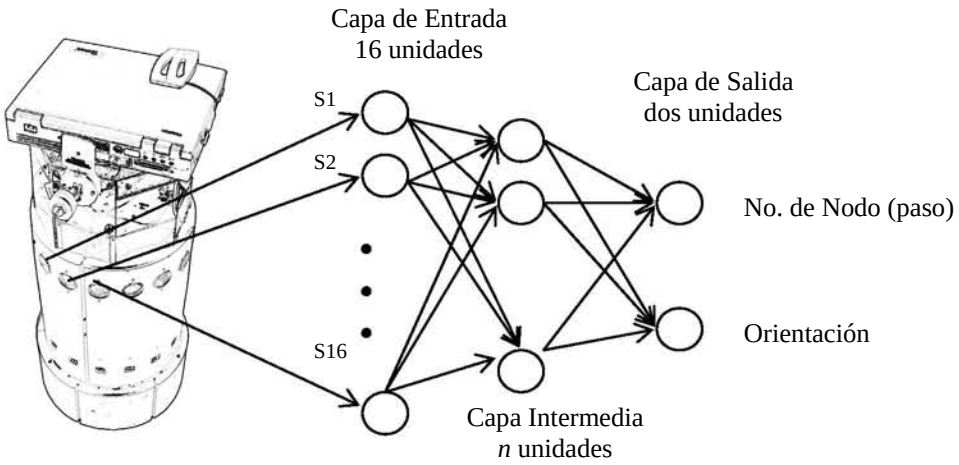

Figura 5.18: Arquitectura propuesta para el problema de auto localización.

Mientras la red neuronal mueve al robot por pasos, en cada paso de avance se toman lecturas con los 16 sonares para diferentes orientaciones del robot (Fig. 5.19). Como el seguimiento de paredes no garantiza visitar la totalidad del espacio navegable, se consideró pertinente que la red no proporcionara directamente como salida la posición (*x*,*y*) en el mapa, ya que, para entrenar a la red, faltaría una gran cantidad de información correspondiente a las zonas libres no visitadas por el robot. En lugar de esto se decidió tomar como ubicación el número de pasos que el robot ha avanzado desde que inició su recorrido hasta el momento en que realizó las lecturas a diferentes orientaciones.

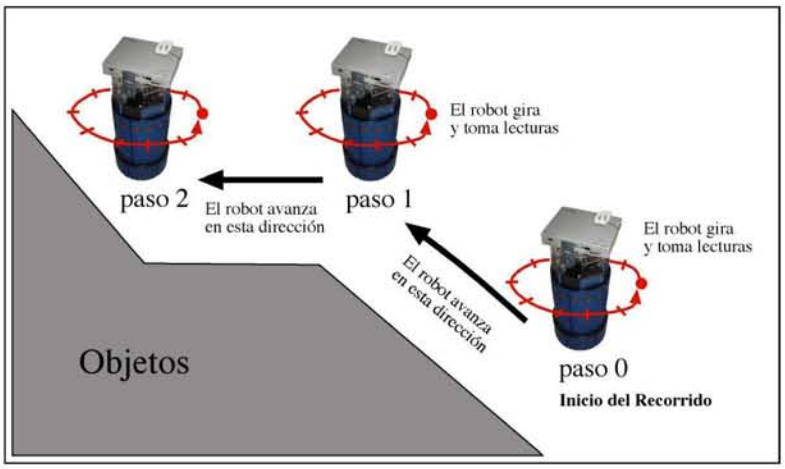

Figura 5.19: Toma de lecturas para el entrenamiento de la red neuronal.

De esta forma se pretende asociar indirectamente cada lectura de los sonares a una posición espacial del robot, en la forma de un número de paso. Evidentemente se deberá guardar registro de la posición espacial del robot durante cada paso de su recorrido para que, una vez alimentada la red y obtenido cierto índice, se pueda conocer la posición final (*x*,*y*) asociada a dicho valor. Por supuesto que la red no podrá proporcionar ubicaciones (índices) que no correspondan con alguna posición previamente visitada por el robot, lo cual suena bastante lógico. El procedimiento anterior evita tener que posicionar manualmente (o hacer que el robot se posicione por sí mismo) en cada ubicación navegable posible del ambiente y tome lecturas en dicha posición.

Se decidió fijar la función de activación de la capa intermedia como tipo sigmoideo (ecuación 3.7) para que proporcionara valores entre 0 y 1, y para la capa de salida una función de tipo lineal ya que este esquema presentó el menor error de entrenamiento.

Cabe destacar que la elección de una arquitectura con sólo dos unidades en la capa de salida no fue al azar. En primera instancia se diseñó una red que tuviera una unidad de salida por cada nodo del recorrido (se probó con 20 posiciones) y una única unidad para la orientación. Al alimentar la red con las 16 lecturas de los sonares, se tomaba el índice de la neurona de salida con la respuesta más alta como el número de nodo. Sin embargo, al tratar de entrenar este tipo de red se observó que, casi todo el tiempo, todas las neuronas de salida (menos la que corresponde al nodo en el que se efectuaron las mediciones) debían presentar una salida baja o igual a cero, y sólo para unos pocos casos de entrenamiento su salida debía ser alta, lo cual evitó que la red pudiera entrenarse.

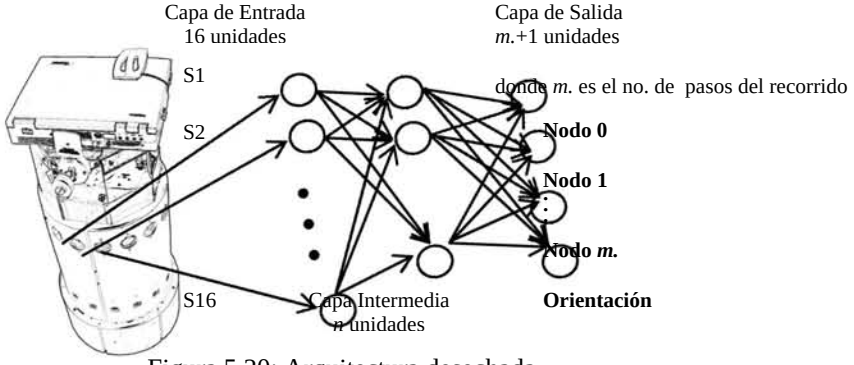

Figura 5.20: Arquitectura desechada.

Supóngase que se tienen 20 pasos (nodos) en el recorrido y el mismo número de neuronas de salida para indicar el nodo o índice. Si por cada paso se hicieron 20 lecturas en diferentes orientaciones, en total se tienen $20 \times 20 = 400$ muestras de entrenamiento. Sin embargo, cada neurona *i* de salida debe proporcionar una salida igual a *cero* para $19 \times 20 = 380$ casos y un *uno* para sólo 20 casos. Lo anterior lleva a la red a presentar una salida baja en todas las neuronas de salida, ya que, como la red fue entrenada mediante el método de retropropagación de errores que minimiza el error cuadrático medio, la red en su conjunto comete menos error si predice ceros exclusivamente, es decir, "aprende" a predecir ceros. Mientras más nodos se tengan (y con ello más neuronas de salida) la proporción de salidas con valor cero se incrementa aún mas, sin importar cuántas lecturas se hagan en cada paso (siempre y cuando sea el mismo para todos los pasos).

Aun si fuese posible entrenar este tipo de red y se deseara agregar un nodo o posición más, sería necesario modificar la arquitectura agregando una unidad más de salida y con ello los conjuntos de entrenamiento y prueba, complicando aún más las cosas.

Por otro lado, se tendría una red con un número elevado de unidades de salida con respecto a las unidades de entrada y el problema de encontrar una heurística para determinar el número de unidades ocultas.

Cabe señalar que el problema de presentar un índice o clase como salida, en función de un vector como entrada se conoce como LVQ (*learning vector quantization*) [Brío 02, pp. 158] y es un problema de aprendizaje supervisado donde. Además de presentar a la red el patrón de entrada durante el entrenamiento, es necesario presentar de igual forma un ***patrón prototipo de la subclase*** a la que pertenece el patrón de entrada, para que finalmente la red aprenda a asociar cada subclase a una clase de salida. Lo anterior implica que por cada conjunto de lecturas de los sonares, se debe proporcionar a la red el patrón prototipo de la subclase correspondiente y saber de antemano cuántas subclases se requieren para realizar correctamente la tarea. Esto, si bien no es imposible, complica enormemente el proceso de entrenamiento ya que se tiene el problema inicial de determinar esos patrones prototipo para un espacio de 16 dimensiones.

Con el esquema propuesto se pretende probar si la misma red es capaz de encontrar dichos patrones prototipo, el número de subclases, asociar cada subclase con un índice que sea presentado a la salida de la red y encontrar la orientación del robot en función de un único patrón presentado a la entrada, lo cual se antoja bastante complicado.

5.8.2 Conjuntos de Entrenamiento y Prueba

Para obtener el conjunto de entrenamiento se modificó el programa para crear el mapa. En cada avance del robot virtual o "paso" se tomaron varias mediciones con los 16 sonares para cada posición (*x*,*y*) visitada a la cual se llamó nodo *i*, girando el robot en cada ocasión sobre su eje 10 grados hasta completar la vuelta completa de 360 grados antes de proseguir su avance a la siguiente posición. En cada punto se almacenó el número del nodo (correspondiente al número de pasos dados desde que comenzó a moverse) y la orientación del robot dentro del ambiente simulado.

S1	S2	S3	S4	S5	S6	...	S12	S13	S14	S15	S16	Nodo	Ángulo
3.5	3.7	2.8	5.8	9.8	4.3	...	2.1	2.8	3.6	8.2	7.6	1	0
1.6	3.4	5.7	8.4	7.2	4.5	...	1.2	1.8	2.0	7.8	6.5	1	45
...	...	...	...	...	...	...	...	...	...	...	...	1	...
3.4	2.3	4.5	2.3	4.5	2.3	...	4.3	5.4	5.1	3.2	4.3	1	270
3.5	3.4	2.3	4.5	3.4	5.6	...	7.8	8.9	7.8	5.6	3.8	1	325
2.3	2.3	2.5	4.5	5.2	7.1	...	8.0	3.2	6.2	8.7	9.8	2	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...
3.4	5.6	2.3	1.2	1.4	1.8	...	2.8	3.7	4.8	9.2	8.7	92	325

Tabla 5.1: Ejemplo de mediciones para un nodo del recorrido.

Se fijó una distancia de avance o paso igual al ancho del robot. En total se obtuvieron 92 nodos siguiendo exclusivamente el contorno de las paredes, con ocho orientaciones por cada uno, es decir, 736 muestras de entrenamiento. Se procedió a utilizar la herramienta Matlab para diseñar una red neuronal que, en función de las 16 entradas constituidas por las mediciones de cada sonar, presentara como salida el número de nodo (del 1 al 92) y el ángulo en el que se encuentra el robot (de 0 a 360 grados). Tanto los datos de entrada (mediciones de los 16 sonares) como los de salida (nodo y ángulo) se normalizaron a valores entre 0 y 1 utilizando la distribución de probabilidad acumulada para realizar la normalización, proceso conocido como *ecualización del histograma* que pretende distribuir las muestras para que su distribución sea lo más uniforme posible y no se hallen en un intervalo muy pequeño lo cual le dificulta a la red establecer las hiper-superficies de separación de clases. Para el caso del número de nodo de salida el valor 0 representa el primer nodo visitado y el 1 el último nodo o 92. Se dividió la unidad en 92 intervalos para determinar el número del nodo de salida. Lo mismo se hizo para normalizar el ángulo de salida entre 0 y 1.

Para obtener el conjunto de prueba se colocó nuevamente el robot virtual en la misma posición de partida que para el conjunto de entrenamiento, sólo que en esta ocasión se colocó desplazado aproximadamente 1/6 del tamaño de la celda (30cm./6 = 5 cm.) a la izquierda de su posición inicial, con la finalidad de poseer lecturas con un grado mínimo de error y poder comparar las predicciones hechas por la red neuronal, en total se obtuvieron nuevamente 736 muestras de prueba.

5.8.3 Número de Parámetros Libres de la Red

Ya que se pretende probar con varias configuraciones de la red, es necesario calcular en cada caso el número de parámetros libres de la misma, para garantizar que existan suficientes datos para poder entrenarla. Para ello se presenta la siguiente tabla:

Capa de entrada	Capa intermedia	Capa de salida	No. Total de Parámetros	No. Muestras	No. Muestras/No. Parámetros	No. Parámetros/No. Muestras
16	4	2	78	736	9.436	0.106
16	8	2	154	736	4.779	0.209
16	16	2	306	736	2.405	0.416
16	24	2	458	736	1.607	0.622
16	32	2	610	736	1.207	0.829

Tabla 5.2: Relación de parámetros libres de la red contra datos de entrenamiento.

Como puede observarse en la Tabla 5.2, para un número de 32 unidades en la capa intermedia, la relación del número total de muestras entre el número de parámetros libres de la red es pequeña. Usualmente se pretende que este factor sea grande para garantizar que cada clase de clasificación se entrene con varias muestras (siempre y cuando los datos de entrenamiento contengan patrones para todas las clases), de no ser así se puede incurrir en un sobreajuste de la red, al existir más parámetros libres que ejemplos de entrenamiento, es decir, la red se ajustará exactamente a los datos de entrada y no generaliza a partir de varias muestras, que es precisamente lo que queremos que haga [Nilsson 01, pp. 48].

5.8.4 Determinación del Número de Unidades Ocultas

Para determinar el número de unidades ocultas se elaboró un programa en Matlab que varía el número de unidades ocultas conforme a nuestras necesidades y obtiene el error mínimo para 100 épocas de entrenamiento. Como sabemos, cuanto más unidades posea una red mayor será el tiempo que tarda en entrenarse y disminuir el error total de la red. Por este motivo se decidió esperar 100 épocas para tomar el error promedio de la red ya que la red con cuatro unidades ocultas entrenaba muy rápido y en cuestión de 15 épocas alcanzaba prácticamente su valor mínimo, mientras que la red con 32 unidades ocultas lo hacía en 25 épocas.

Los mejores resultados en el error de entrenamiento se obtuvieron con la red de 32 unidades en la capa oculta, lo cual puede suponer que, al existir un mayor número de parámetros libres en la red, ésta se ajusta mejor a los datos de entrenamiento.

5.8.5 Entrenamiento de la Red Neuronal

Una vez determinado el número de unidades de la capa oculta se procedió a entrenar la red en Matlab utilizando para ello retropropagación de errores y como medida de error el error cuadrático medio de la red.

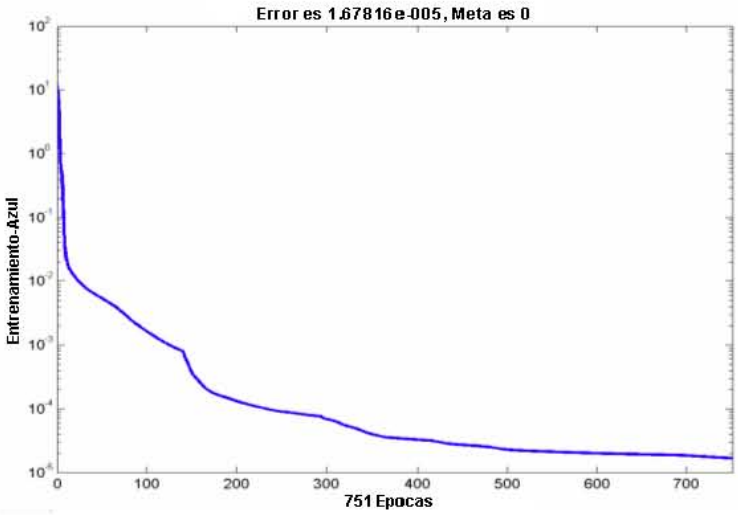

Figura 5.21: Error de entrenamiento de la red neuronal.

Cabe señalar que en esta ocasión dejamos que la red se entrenara hasta un número elevado de iteraciones (750) para observar el comportamiento del error de entrenamiento.

Como puede apreciarse en la Figura 5.21, el error de entrenamiento presenta una disminución casi constante entre 50 y 150 épocas, a partir de este punto vuelve a presentar una caída abrupta que se mantiene prácticamente constante entre 200 y 300 épocas, para finalmente sufrir otro ligero descenso entre 300 y 350 épocas. Se decidió tomar 250 épocas como un buen punto para detener el entrenamiento de la red ya que es la primera ocasión donde el error se mantiene prácticamente constante.

No nos sorprende que la red haya podido "aprender" a calcular el nodo y orientación de salida en función de una única observación de sus sonares, ya que toda arquitectura neuronal se puede ajustar perfectamente a los datos de entrada (con un número suficiente de parámetros). Muestra de ello es el error mínimo que se obtiene al entrenar con 250 épocas que es de aproximadamente un 0.1 %.

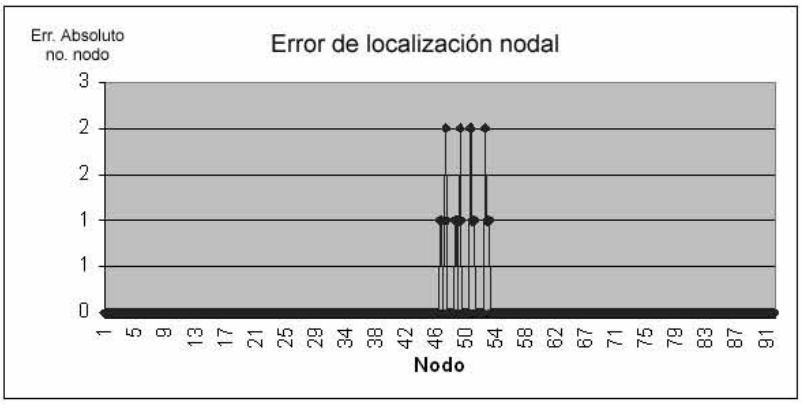

Figura 5.22: Error de localización nodal para el conjunto de entrenamiento (250 épocas).

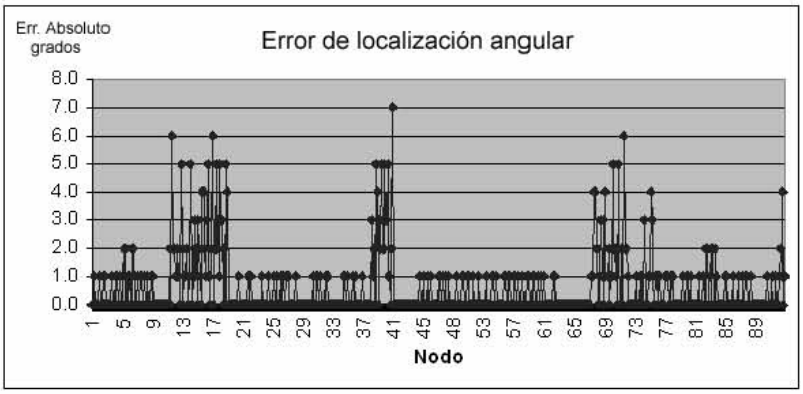

Figura 5.23: Error de localización angular para el conjunto de entrenamiento (250 épocas).

Como puede apreciarse en las Figuras 5.22 y 5.23 la localización con el conjunto de entrenamiento es bastante precisa, únicamente se observa una pequeña "confusión" en el número de nodo en la parte central del recorrido (debido a la simetría del espacio de trabajo), mientras que el error en la localización angular apenas llega a siete grados en su caso más extremo. A continuación se presentan los resultados con el conjunto de prueba.

5.8.6 Prueba de la Red Neuronal

Como se explicó en la sección 5.8.2, como conjunto de prueba se alimentaron a la red las mediciones hechas por el robot virtual siguiendo exactamente el mismo recorrido sobre el ambiente que para el conjunto de entrenamiento, sólo que en este caso, se desplazó ligeramente hacia la izquierda la posición de partida del robot. A continuación se presentan los resultados obtenidos.

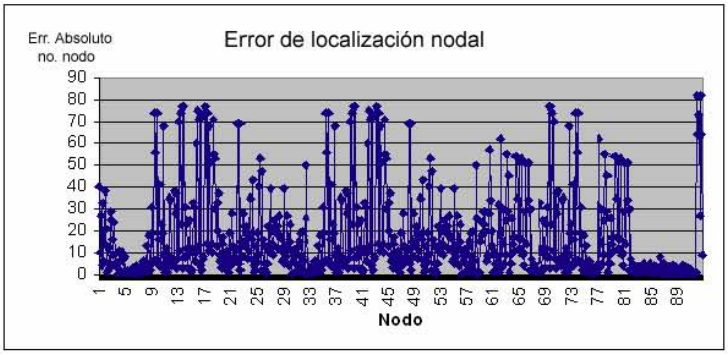

Figura 5.24: Error de localización nodal para el conjunto de prueba con desplazamiento.

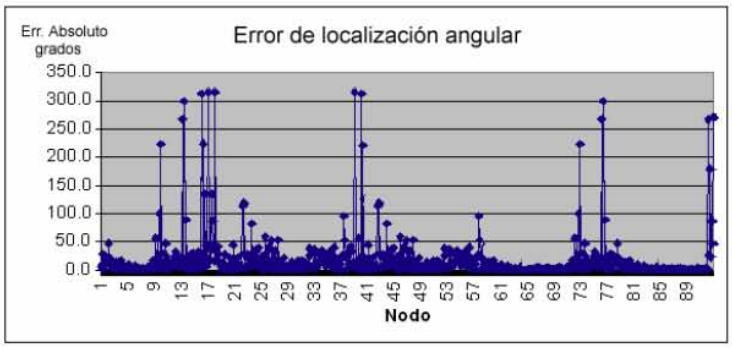

Figura 5.25: Error de localización angular para el conjunto de prueba con desplazamiento.

Como puede observarse en las Figuras 5.24 y 5.25, cuando se introdujo una ligera variación en los datos de entrada la red se equivoca rotundamente en cierto intervalo y en el resto, no obstante el error no es tan grande (de cinco a ocho nodos y abajo de 15 grados) es suficiente para producir un error grande en la localización.

5.9 Adición de Errores de Movimiento y Lectura en el Ambiente Virtual

Una vez completados los procedimientos de movimiento autónomo y creación del mapa en el simulador, se agregó ruido gaussiano a los sonares (media 5 cm. y varianza 3 cm.) directamente en el simulador, y se ejecutó nuevamente la rutina de creación del mapa.

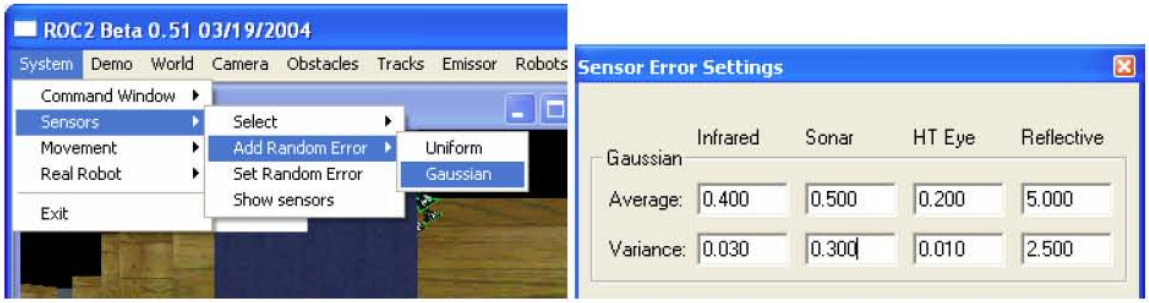

Figura 5.26: Adición de error gaussiano a las mediciones con los sonares virtuales.

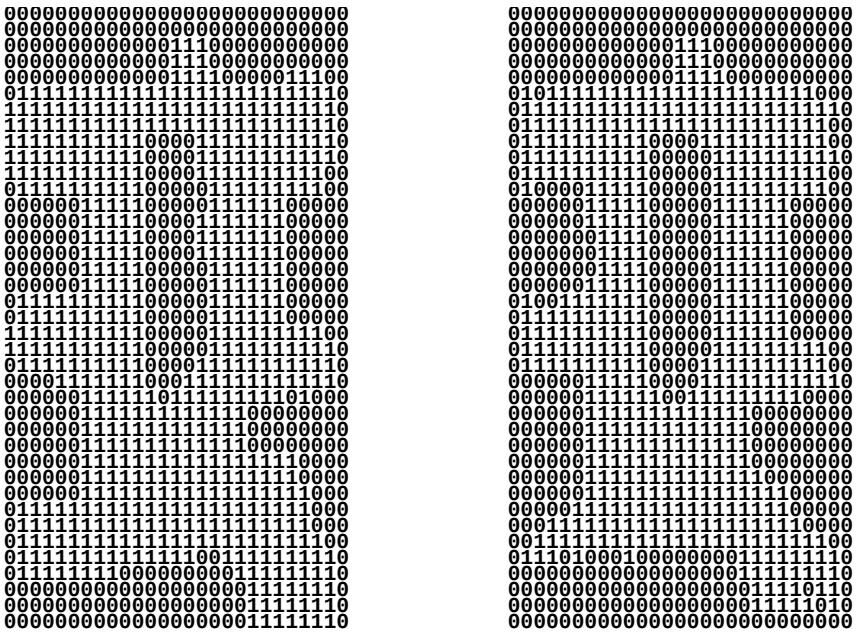

Figura 5.27: Mapa sin ruido (izquierda) y mapa elaborado con ruido gaussiano (derecha).

Como puede apreciarse en la figura anterior el mapa final prácticamente no resultó afectado por los errores en las mediciones de los sonares, en gran medida debido a que cada celda mide 30 × 30 cm. y un error de sólo 5 cm. no resultó significativo. A continuación aumentamos el ruido gaussiano con una media de 0 cm. y varianza de 20 cm. y enseguida con media 0 cm. y varianza de 50 cm. (se eligió una media de 0 para que existiesen errores de medición tanto positivos como negativos).

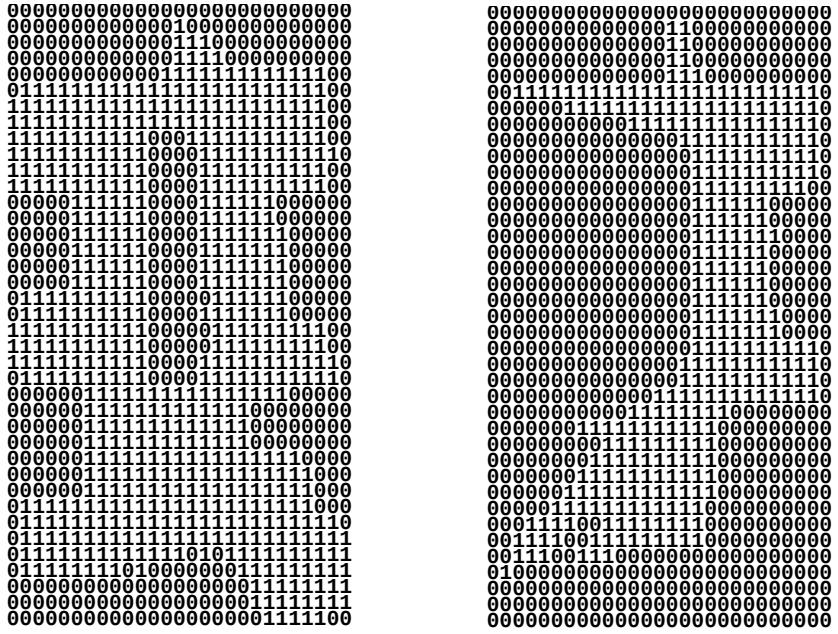

Figura 5.28: Mapa con ruido $\overline{X}$ = 0, σ = 20 cm. (izquierda) y $\overline{X}$ = 0 cm., σ = 50 cm. (derecha).

Cabe aclarar que en el segundo caso (con una varianza de 50 cm.) el tiempo que tarda el robot en crear el mapa se incrementa considerablemente (casi al triple) ya que los errores en los sonares utilizados también para el algoritmo de movimiento autónomo en ocasionas evitan que siga fielmente a las paredes y dé muchas vueltas inútiles, también dichos errores evitan que determine correctamente que existe una separación suficiente entre ciertos objetos y evite pasar entre ellos, dejando zonas sin visitar.

De igual forma que para el caso de los sonares, es posible agregar ruido a los movimientos del robot, tanto al desplazamiento como a los giros.

Figura 5.29: Adición de error gaussiano a los movimientos del robot virtual.

En esta ocasión agregamos ruido gaussiano con media de 5 cm. y varianza de 3 cm. a los desplazamientos y ruido gaussiano con media de 1 grado y varianza de ½ grado a la rotación del robot (Fig. 5.30 izquierda). Enseguida se modificó a media 0 y varianza de 10 cm. para los desplazamientos y media 0 y varianza de 5 grados (Fig. 5.30 derecha).

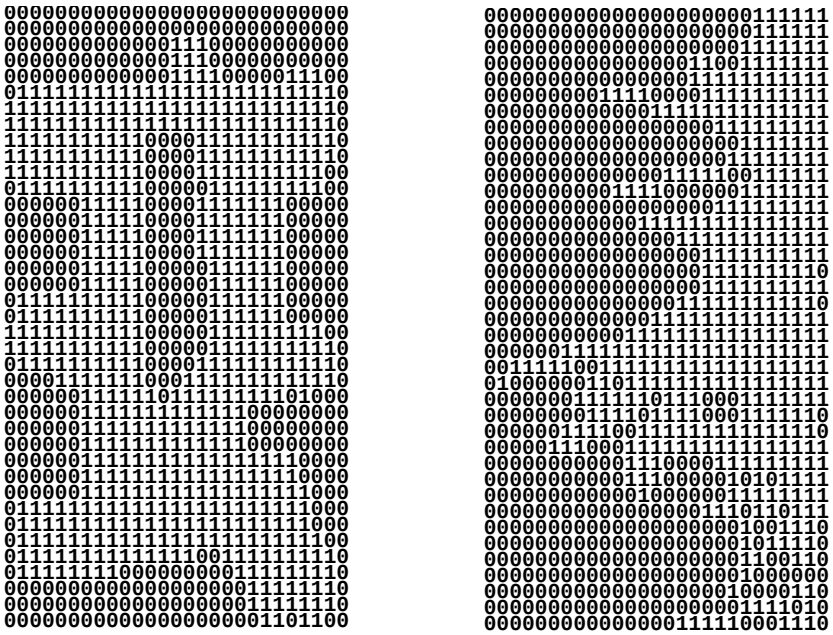

Figura 5.30: Mapas con errores de movimiento.

De nueva cuenta se puede apreciar que los algoritmos de creación del mapa no se ven afectados por errores pequeños en los movimientos del robot. Si estos errores llegan a

alcanzar los valores que se han probado para la Fig. 5.30, puede notarse que su efecto en el mapa resulta más perjudicial que los errores en las mediciones de los sonares.

El efecto más grave de los errores de movimiento lo constituyen sin duda los errores en los giros, ya que afectan directamente al algoritmo de movimiento autónomo y hacen que la orientación tomada para detectar los objetos del mapa sea equívoca, por consiguiente los objetos son localizados en posiciones erróneas. Por otro lado, los errores en el movimiento pueden hacer que el robot colisione con su medio ambiente y quede atascado (Fig. 5.29 derecha).

Como el caso de la creación del mapa, se probó nuevamente la red neuronal implementada para la localización. Se partió exactamente del mismo punto que para el conjunto de entrenamiento, pero en esta ocasión agregamos ruido gaussiano a las mediciones de los sonares (media: 5 cm., desv. estándar: 3 cm.).

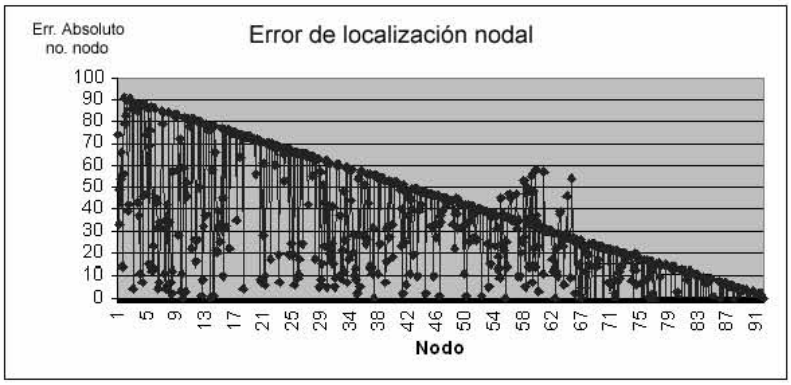

Figura 5.31: Error de localización nodal para el conjunto de prueba con error gaussiano.

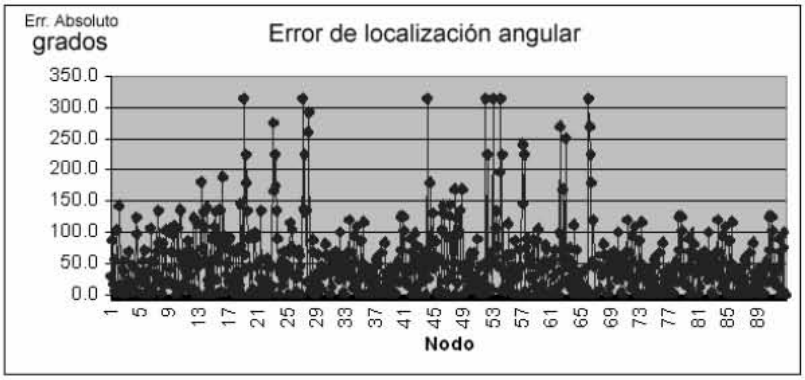

Figura 5.32: Error de localización angular para el conjunto de prueba con error gaussiano.

Cuando se inducen errores en los sonares (Figuras 5.31 y 5.32) el error de localización es aún mayor que cuando se desplaza al robot ligeramente de las posiciones entrenadas, y con ello la red se confunde completamente.

Con base en los resultados obtenidos para la red neuronal en la tarea de localización, es posible afirmar que no obstante que este método permite relacionar con bastante exactitud las lecturas de los sonares con el número de nodo y orientación del robot para el conjunto de entrenamiento, resulta sumamente sensible a ligeras variaciones en los datos de entrada.

Es posible pensar que existió un sobreajuste de la red, ya que la relación datos de entrada-parámetros es casi uno a uno; sin embargo, para que el método pueda utilizarse confiablemente, éste debe proporcionar un error de localización realmente mínimo y, si se disminuye el número de épocas de entrenamiento y se mejora con ello el desempeño global de la red para el conjunto de prueba, el grado de error que se requiere de la red debe ser similar al obtenido con el conjunto de entrenamiento en 250 épocas, es decir realmente mínimo.

De acuerdo con los resultados obtenidos, se puede concluir que la arquitectura de red elegida no resultó adecuada para los propósitos buscados debido a las alteraciones producidas por el más mínimo ruido.

5.10 Pruebas en el Ambiente Real de Operación

5.10.1 Resultados de las Pruebas

Para poder probar los algoritmos de movimiento autónomo y creación del mapa en el ambiente real (Laboratorio de Biorrobótica) se elaboró un programa en C++ que se comunica con el robot real de forma inalámbrica. El formato de los comandos que recibe el robot real es la misma que la utilizada en el simulador y los datos recibidos desde el robot real poseen la misma estructura y unidades que las utilizadas en el simulador.

Como punto de partida en el ambiente real se eligió un sitio al centro de la zona más amplia del mapa que se encuentra libre. El robot se orientó al norte con respecto al mapa del laboratorio y se ejecutó la rutina de creación del mapa y movimiento autónomo del robot, en esta ocasión con lecturas reales.

Al tratar de probar los mismos algoritmos de creación del mapa que en el ambiente virtual, se presentó el primer problema serio de la investigación. Al observar el movimiento realizado por el robot para seguir paredes (en este caso escritorios y mesas) se observó que no pudo realizar ningún recorrido correctamente ya que en todas las ocasiones chocó contra alguna mesa o escritorio casi inmediatamente después de iniciar el proceso de navegación autónoma. Como máximo pudo avanzar una distancia de 3 m.

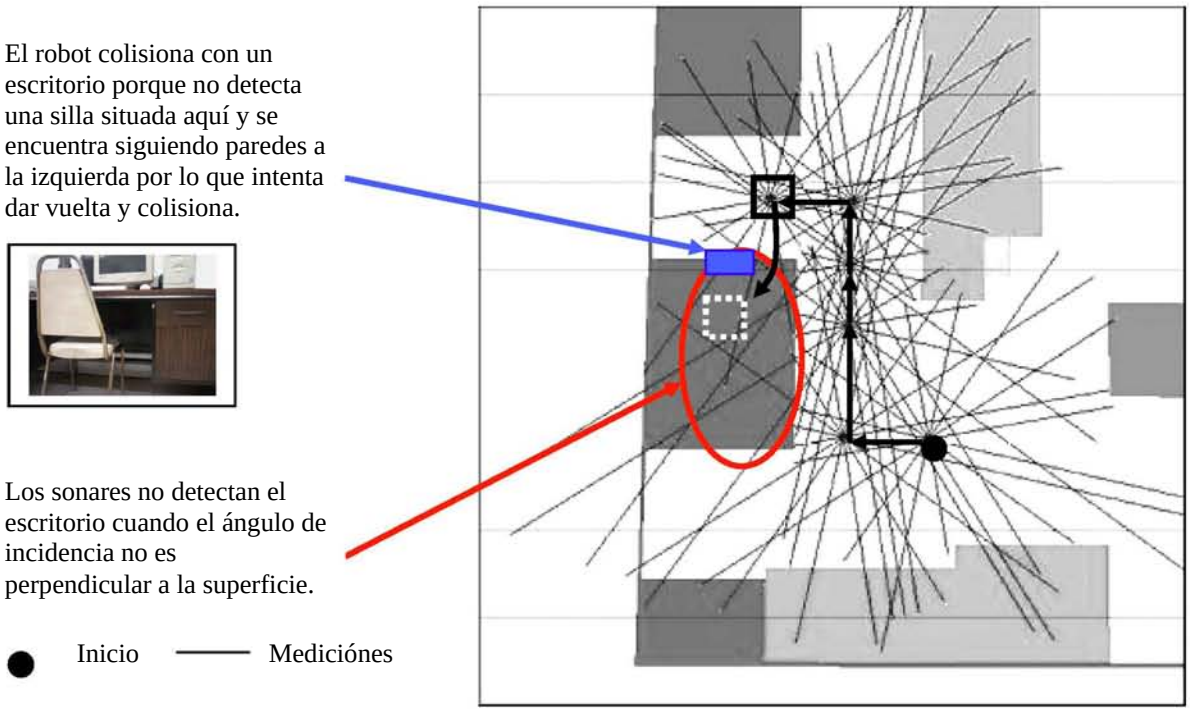

Figura 5.33: Navegación usando algoritmo de seguimiento de paredes con lecturas reales. Las líneas muestran las mediciones hechas por los sonares en cada dirección.

Como se muestra en la figura anterior, de alguna forma los sonares "penetran" en las estructuras (que principalmente son escritorios de madera, sillas, mesas con sus espacios abiertos cubiertos con cartón y cajas de cartón) lo que provoca que se generen lecturas erróneas del ambiente. La magnitud del error resulta bastante elevada si la comparamos con la que simulamos en el ambiente virtual.

Al efectuar un análisis a profundidad sobre los primeros datos obtenidos, fue posible determinar que el error obtenido en las mediciones de los sonares no presenta una distribución gaussiana, como la simulada en el ambiente virtual, sino que este error depende enormemente del ángulo de incidencia de la onda del sonar sobre la superficie reflectora y del tipo de material de la misma. Por este motivo el robot real colisiona con los objetos ya que en muchas ocasiones no los percibe. Es posible asegurar que los sonares utilizados tuvieron un bajo desempeño en el ambiente real de operación, por causa del error intrínseco a las mediciones que superó los límites simulados.

Por este motivo se decidió realizar una evaluación del desempeño físico de los sonares del T×8, realizando para ello distintas mediciones con diferentes materiales con la finalidad observar el desempeño de los sonares del robot real y su rango de error y precisión en sus lecturas.

5.10.2 Evaluación del Desempeño de los Sonares del *T×8*

El robot T×8 posee un anillo de 16 sonares distribuidos de manera uniforme a lo largo del perímetro del robot. A continuación se muestra un diagrama de su ubicación:

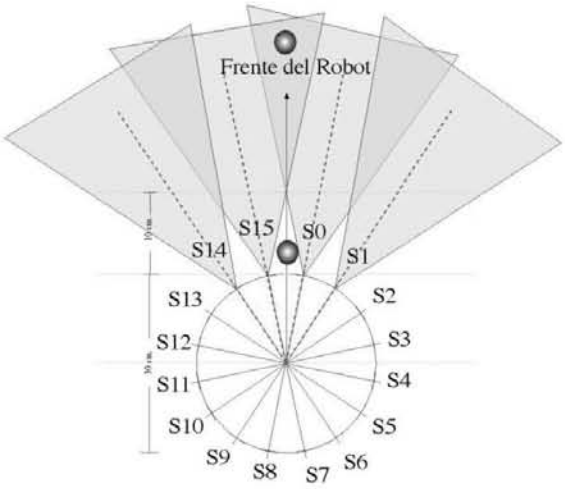

Figura 5.34 Distribución de los sonares del robot T×8, con un objeto en zona "ciega" y otro en zona "visible".

Inicialmente se colocó una lámina metálica perpendicular a la línea de emisión de uno de los sonares del robot y se realizaron distintas mediciones, modificando el ángulo de incidencia de la señal sobre el objeto, para evaluar la precisión de los sonares (Fig. 5.36). Cabe señalar que, como se indica en la Figura 5.34, si el objeto se encuentra entre dos sonares a menos de 10 cm. de distancia del sonar, éstos no lo detectan. Si el objeto se encuentra a menos de 10 cm. frente al sonar, la medición mínima es 10 cm.

Objeto:	Lamina Metálica	
D real (cm.)	*D* medida	Error Abs. (cm.)
0	9.2	9.2
10	10.3	0.3
30	30.2	0.2
60	60.4	0.4
100	100.21	0.21

Tabla 5.3: Mediciones con el sonar del T×8 para una lámina metálica situada perpendicularmente.

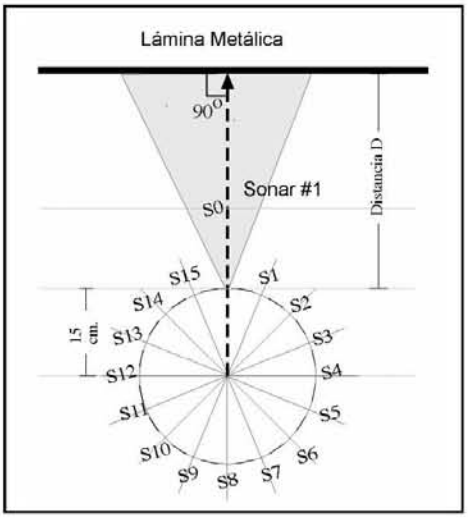

Figura 5.35: Colocación del robot y objeto, para la prueba del desempeño de los sonares.

Como puede observarse el error obtenido en las mediciones de los sonares es pequeño para el objeto medido. A continuación efectuó otra medición con el mismo objeto, pero en esta ocasión variando el ángulo de incidencia de las emisiones del sonar.

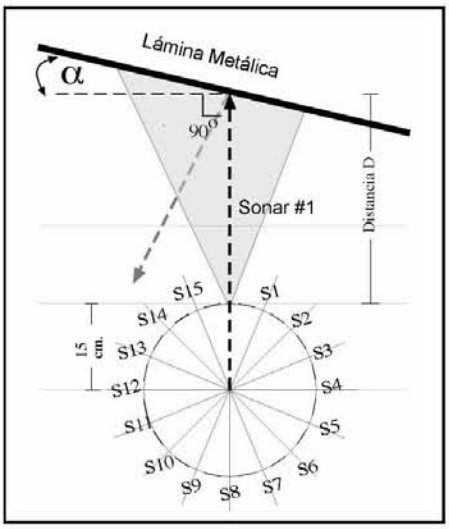

Figura 5.36: Colocación del robot y objeto, para la prueba variando el ángulo de incidencia con respecto a la perpendicular.

Objeto:	Lamina	Metálica	
D real (cm.)	α (grados)	*D* medida	%error
50	0	50.3	0.6
50	15	50.2	0.4
50	30	50.4	0.8
50	45	51.5	3
50	60	62	24
50	75	91	82

Tabla 5.4: Mediciones con el sonar del T×8 para variaciones con respecto a la perpendicular.

Se observó que a partir de los 45 grados de desviación de la perpendicular (renglón marcado en la Tabla 5.4), las lecturas proporcionadas por el sonar evaluado comenzaron a oscilar notablemente (por encima del valor real y variando hasta en un 100% con respecto a su valor real) por lo que por encima de ese valor de desviación no se pueden considerar dichas mediciones estables ni confiables.

A continuación se repitió el experimento anterior, para distintos tipos de materiales como cartón y tela (mantel de poliéster).

Caja de Carton			
D real (cm.)	α (grados)	*D* medida	%error
50	0	50.1	0.2
50	15	153	206
50	30	163	226
50	45	162	224
50	60	173	246
50	75	222	344

Tabla 5.5: Mediciones con el sonar del T×8 para una caja de cartón corrugado.

Tela delgada			
D real (cm.)	α (grados)	*D* medida	%error
50	0	49.9	0.2
50	15	104.3	108.6
50	30	163	226
50	45	162	224
50	60	173	246
50	75	222	344

Tabla 5.6: Mediciones con el sonar del T×8 para tela delgada (poliéster).

Para estos materiales una ligera variación del ángulo con respecto a la perpendicular significa que las ondas son absorbidas por el material (para el caso de la caja de cartón) o que dichas señales lo atraviesan (para el caso de la tela). Sin embargo, es de hacer notar que en ambos casos cuando no existe desviación de la perpendicular el sonar mide correctamente la distancia al objeto sin importar si se trata de metal, cartón o tela.

Cuando se rebasa la desviación máxima permitida para cada material, las mediciones comienzan a oscilar por encima del valor real (por ejemplo para la caja de cartón a los 45 grados las mediciones eran sucesivamente y en ciclos para el mismo sonar: 182, 237, 325 y 558 cm.) lo cual puede presentar objetos "fantasma" debidos a mediciones erróneas si todas las mediciones se consideran como válidas.

Finalmente, para descartar un posible problema del sonar utilizado en las pruebas, se realizaron varias mediciones (300) en cada uno de los 16 sonares del robot, situando una lámina metálica a una distancia constante de 100 cm., en forma perpendicular a la señal de cada sonar evaluado, es decir, con un ángulo de desviación α de la perpendicular igual a *cero* (el cual sabemos no presenta errores en la medición).

Sonar	1	2	3	4	5	6	7	8
Media (cm.)	100.21	99.82	99.55	99.33	99.72	99.92	100.10	100.37
Desv.est. (cm.)	0.03	0.05	0.06	0.05	0.04	0.07	0.02	0.14
Media Sonar - Media de las Medias (cm.)	0.23	0.62	0.89	1.11	0.72	0.52	0.34	0.07

Sonar	9	10	11	12	13	14	15	16
Media (cm.)	100.30	100.72	101.01	101.01	101.45	101.47	101.30	100.82
Desv.est. (cm.)	0.02	0.04	0.06	116.10	0.05	0.06	0.07	0.07
Media Sonar - Media de las Medias (cm.)	0.14	0.28	0.57	0.57	1.01	1.03	0.86	0.38

Media de las Medias: 100.44

Tabla 5.7: Comparación de mediciones con los 16 sonares del T×8.

Como puede observarse en la tabla anterior, en realidad existe poca diferencia entre las mediciones de los 16 sonares (1.11 cm.) como máxima diferencia entre la media de todas las medias y la media de un sonar en particular lo cual resulta despreciable. Con base en las mediciones anteriores es posible afirmar que el comportamiento observado para el sonar evaluado será el mismo para el resto de los sonares.

Sin embargo, dependiendo del material de que se trate, una ligera variación en el ángulo de incidencia hace que el robot deje de percibir el objeto en forma correcta. Por este motivo en el ambiente real no funcionó correctamente el algoritmo de seguimiento de paredes, el cual calcula la distancia libre a partir de las mediciones de los sonares. Para mejorar el seguimiento de paredes se debe encontrar alguna forma de relacionar las mediciones de los sonares que le permita no ser susceptible a los errores en dichas mediciones. Sin embargo, dicha relación no es clara y requiere de tiempo para analizarse e implementarse.

Capítulo 6

Aplicación de Técnicas Neuronales

6.1 Red Neuronal para el Movimiento Autónomo

Para evitar tiempos de análisis, modificación y prueba del algoritmo de movimiento autónomo en el ambiente real de operación, se decidió utilizar las capacidades de las redes neuronales. Se decidió diseñar y entrenar una red neuronal que siguiera las paredes y que encontrase, de manera práctica, una función que tomara en cuenta las 16 mediciones de los sonares simultáneamente, encontrara la relación entre las mismas y determinara la dirección final de movimiento del robot.

6.1.1 Diseño de la Arquitectura

Se probaron dos arquitecturas: una arquitectura que con los valores de los 16 sonares como entradas, proporcionara una salida que indicara el ángulo en el cual el robot se debería a desplazar (de 0 a 359 grados) y otra que tuviera 16 neuronas de entrada, un número de neuronas por definir en la capa intermedia y ocho neuronas en la salida (utilizando 8 direcciones de movimiento del robot: norte, sur, este, oeste, noroeste, noreste, suroeste y sureste), tomando como salida (dirección de movimiento) aquella dirección perteneciente a la neurona que entregara el máximo valor de salida entre las ocho posibles.

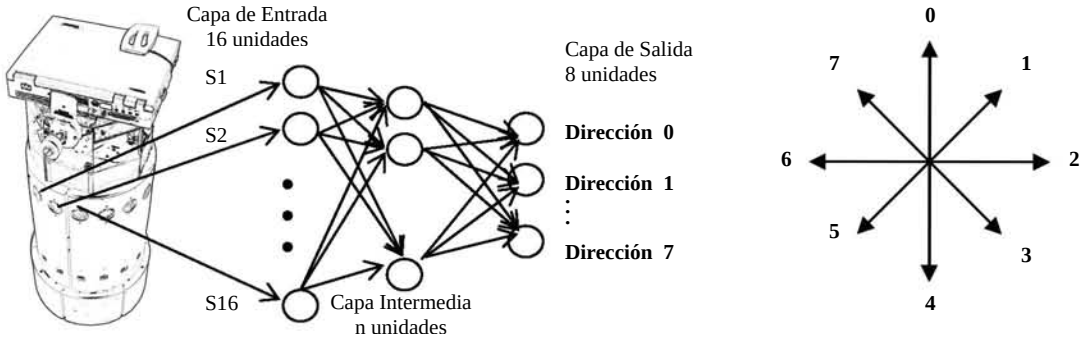

Figura 6.1: Arquitectura para la red neuronal de seguimiento de paredes y las 8 direcciones de movimiento.

Luego de algunas pruebas preliminares con datos tomados del simulador, se obtuvo un mayor error de movimiento al cargar toda la decisión de la dirección final de movimiento a una sola neurona, operando en un rango de 0 a 1 (neurona sigmoidea). Por este motivo se utilizó finalmente la arquitectura con 16 neuronas en capa de entrada y ocho de salida de tipo sigmoidal (ec. 3.7) indicando la dirección de movimiento del robot.

6.1.2 Conjuntos de Entrenamiento y Prueba

Para generar el conjunto de entrenamiento se fue moviendo al robot por pasos de avance de 30 cm. Una rutina realiza una medición con los 16 sonares, los almacena y solicita la dirección del siguiente movimiento del robot (de 0 a 7). Para proporcionar la dirección de movimiento del robot se utilizó una palanca de juegos (joystick, Fig. 6.2).

Figura 6.2: Obtención del conjunto de entrenamiento.

De esta forma por cada punto del recorrido se almacena en un archivo las entradas a la red (mediciones de los 16 sonares en ese punto) y como salida la dirección de movimiento del robot (valor 1 a 8).

En total se realizaron ocho recorridos de entrenamiento (moviendo al robot por todo el ambiente de trabajo) partiendo de diferentes puntos y orientaciones dentro del mapa. El número total de vectores de entrenamiento fue de 960, destinando el 80% para datos de entrenamiento y el resto como datos de prueba. Se entrenó al robot moviéndose por pasillos con escritorios, sillas e inclusive se hicieron algunos entrenamientos utilizando láminas metálicas delimitando las áreas navegables.

Para entrenar y utilizar la red neuronal se implementó en lenguaje C++ una rutina para crear la red neuronal, leer los datos de entrenamiento y prueba, normalizar las entradas y salidas, entrenar la red mediante el algoritmo de retropropagación y para probar la red alimentando las entradas y obtener una salida de inmediato.

Como el proceso de prueba resulta sumamente sencillo en este caso, pues basta con sólo colocar al robot en alguna posición de arranque, hacer que obtenga una muestra con sus sonares y determine la dirección de avance por sí sólo, es posible observar directamente el comportamiento de la red puesta en operación. Una vez que se encontró el número de épocas de entrenamiento que proporcionaba el menor error de prueba, se posicionó al robot real en una ubicación de partida cercana a alguna de las previamente entrenadas, con la misma orientación que en el entrenamiento, y se determinó experimentalmente al igual que con el ambiente simulado si la red aprendió suficientemente bien y si le afectaron significativamente las variaciones en la mediciones de los sonares. Lo anterior puede considerarse erróneo ya que podría llevar a preferir una red que presentara un mayor error de prueba que otra. Sin embargo, el interés no se centró exclusivamente en el error promedio de la red, sino en los resultados asociados al error cometido, es decir, se poseen varias métricas de desempeño. Así, una red que moviera al robot un poco más lejos de las paredes que en el recorrido entrenado y presentaría un mayor error promedio (en caso de elegir incorrectamente un avance que lleve al robot directamente hacia una pared), resulta menos riesgosa que otra red que lo lleva muy próximo a las paredes en casi todo el recorrido (digamos 99% de las veces) y en una sola ocasión falla y lo lleva a estrellarse de inmediato con un objeto o muro.

Cabe aclarar que el método de evaluación anterior no sólo sirvió para determinar el número de unidades en la capa oculta. También se utilizó para determinar el tipo de normalización aplicada a los datos de entrada de la red, ya que se probó con la ecualización del histograma, con la discriminación por umbral de las entradas (ponerlas en 0 o 1 exclusivamente, si la medición superaba cierto límite) y con la utilización de los datos en bruto (sin normalizar) directamente entregados por los sonares. Por último en cuanto a las épocas de entrenamiento se probó con 100 y 200 épocas y se evaluaron los resultados. El procedimiento de evaluación anterior se hizo tomando en cuenta qué tan sensible resultaba cada red a los obstáculos y que el movimiento final del robot realizara el objetivo final de seguimiento de contornos para el cual la red fue entrenada. Por observación se determinó si hacía falta algún tipo de entrenamiento en específico (como por ejemplo con cajas de cartón) hasta que el robot navegara razonablemente bien.

La red final es una red homogénea de tipo sigmoideo con 16 unidades en la capa de entrada, 16 unidades ocultas y ocho unidades de salida, dicha red posee 416 parámetros libres por lo que las 960 muestras resultan adecuadas para el entrenamiento.

Se utilizaron 200 épocas de entrenamiento para la red neuronal (utilizando un factor de momento de 0.1 y deteniendo el entrenamiento al alcanzar un gradiente mínimo de 0.0001), entregando un error del 18% para el conjunto de prueba (que se puede considerar malo, sin embargo en la práctica resultó bastante eficaz). A los datos de entrada se les aplicó una ecualización del histograma que los deja en el rango de 0 a 1 en función de su densidad de probabilidad acumulada, ya que la red resultaba mas sensible a los obstáculos que simplemente especificando un umbral o inclusive mediante normalización que, no obstante permitían seguir mejor los contornos, no eran muy sensible a las esquinas de los escritorios y el robot colisionaba con ellas en lugar de rodearlas.

Como era de esperarse, la red nunca funcionó bien utilizando los datos de entrada en crudo, ya que en ocasiones las variaciones en las mediciones de los sonares reales son del orden de metros (inclusive llegamos a obtener lecturas de 350 metros).

6.1.3 Evaluación del Movimiento Autónomo del Robot

Una vez entrenada y probada la red neuronal de movimiento autónomo se colocó en un punto arbitrario (con una orientación favorable), se le indicó su posición inicial de (24.0 ,18.0) mirando a 0 grados y se ejecutó la rutina de movimiento autónomo. Se trataron de cubrir los espacios abiertos debajo de las mesas con cajas de cartón y en el hueco de los escritorios se colocaron sillas, esto con la finalidad de evitar que el robot tratase de introducirse debajo de ellos. A continuación se muestran los recorridos autónomos efectuados por el robot:

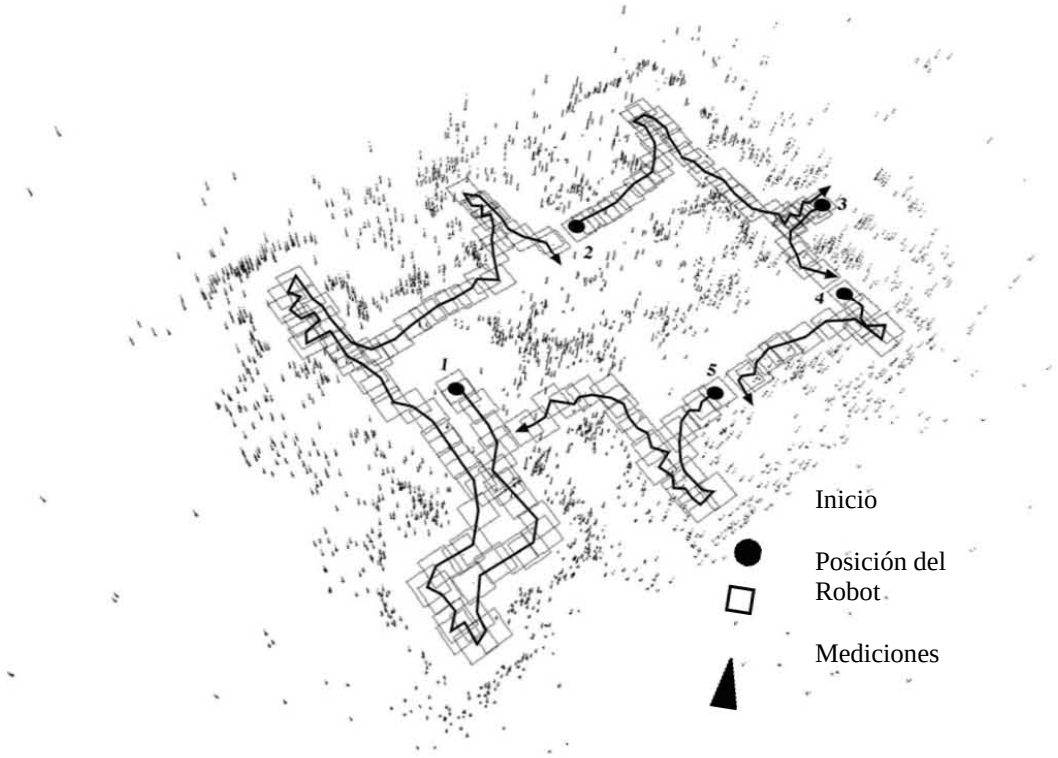

Figura 6.3: Movimiento autónomo del robot con red neuronal y mediciones de los sonares llevadas a puntos sobre el plano *xy*. Los números indican los recorridos efectuados.

Como puede observarse en la Figura 6.3, el robot no fue capaz de seguir por completo el contorno de los objetos, ya que en cuatro ocasiones colisionó con las cajas de cartón o se trataba de introducir en las aberturas de los escritorios. Fue necesario interrumpir la toma de muestras y se colocó al robot en una posición cercana al sitio donde colisionó, indicándole al robot en todos los casos su nueva posición y orientación de arranque. El recorrido se detuvo manualmente cuando hubo dado la vuelta al laboratorio y regresó a una posición previamente visitada, al final del 5º recorrido.

Si bien durante el entrenamiento se colocaron láminas metálicas, se evitó utilizarlas en el ambiente real, ya que el objetivo era tratar de generar un mapa sin modificar el ambiente substancialmente. Como resultado de que le fue informada al robot su nueva posición y orientación luego de cada colisión, no se observa un alto grado de error en los movimientos. Sin embargo, es necesario probar con un recorrido sin interrupciones.

Es posible afirmar que el uso de la red neuronal para implementar el movimiento autónomo del robot funcionó mucho mejor que el algoritmo determinístico implementado en el simulador, el cual no permitió al robot avanzar por más de tres metros.

6.1.4 Creación del Mapa

Una vez que se recabaron las muestras dentro del ambiente real, se ejecutó la rutina de creación del mapa implementada para el ambiente virtual con los siguientes resultados:

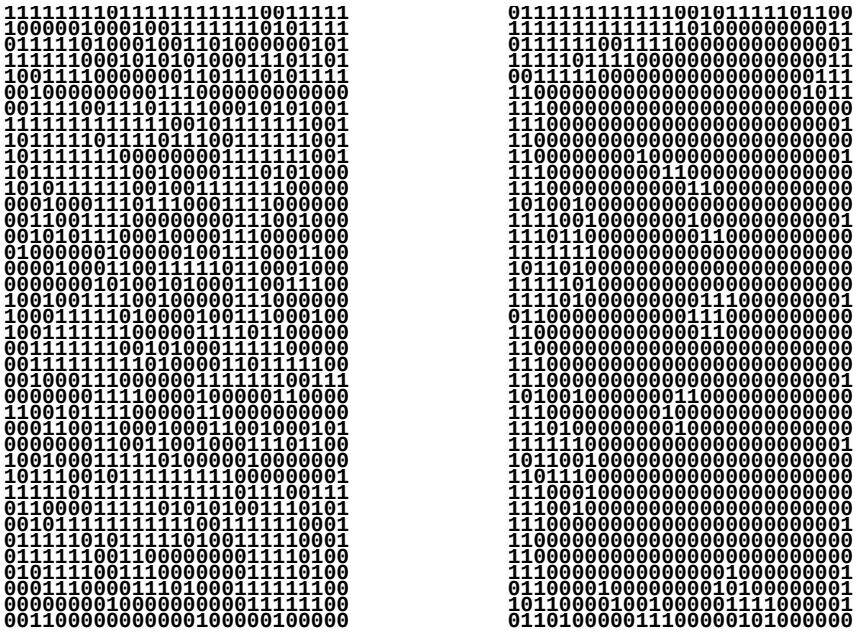

Figura 6.4: Mapa real bloqueando puntos localizados por los sonares (izquierda) y mapa real utilizando el algoritmo de “rasurado” (derecha).

Como puede observarse en la figura anterior, el resultado es desastroso ya que, debido a los errores en las mediciones proporcionadas por los sonares al incidir en ángulos no perpendiculares sobre las superficies del entorno, el algoritmo “limpia” el mapa mucho más de la cuenta, en el caso de usar el “rasurado”. Cuando utilizamos el algoritmo que bloquea la celda al extremo de la línea que corresponde a la medición del sonar se posicionan objetos inexistentes en el ambiente real y queda muy poco espacio navegable o se vuelve prácticamente imposible navegar. Por tal motivo se deberá buscar algún método de filtrar las mediciones erróneas o implementar algún otro que sea menos sensible a los errores cometidos por la naturaleza intrínseca de los sonares.

6.1.5 Auto Localización

Con los datos obtenidos durante el recorrido del robot en el ambiente real se entrenó nuevamente la red en Matlab, ahora introduciendo con ello el ruido intrínseco de los sonares al proceso de entrenamiento de la red. La arquitectura se conservó intacta para evaluar la diferencia. Se colocó nuevamente al robot en una ubicación previamente visitada y se tomó un conjunto de lecturas con los 16 sonares. Los resultados, como era de esperarse, arrojaron un error realmente grande de localización al existir variaciones en las mediciones de los sonares aún mayores que las probadas en el ambiente virtual.

6.1.6 Resultados en el Ambiente Real de Operación

Como pudo observarse, debido a las variaciones en las mediciones de los sonares en el ambiente real, tanto los algoritmos de movimiento autónomo determinístico como los de creación del mapa tuvieron un mal desempeño, en este punto se decidió explorar al igual que para la navegación autónoma la utilización de técnicas con redes neuronales artificiales como alternativa a los métodos anteriores, con la finalidad de observar su desempeño con las limitaciones impuestas por las características de los sonares.

En cuanto a la auto localización es posible atribuir los pobres resultados obtenidos a que se utilizaron exclusivamente las 16 entradas de los sonares para determinar el nodo y la orientación, lo cual representa una complejidad alta ya que la red debe trabajar con un espacio de 16 dimensiones y proporcionar una salida de sólo dos (nodo y orientación).

6.2 Red de Agrupamiento

Como el robot por sí mismo no fue capaz de completar el recorrido por el ambiente real sin modificarlo substancialmente y es importante tener una toma continua de muestras, utilizando exclusivamente como medio de localización la odometría del robot, se realizó un recorrido guiado (utilizando el joystick) para evitar que el robot colisionara con los objetos y se interrumpiese la toma de muestras, alterando con ello la ubicación del robot.

En este punto se utilizaron las 16 mediciones de los sonares y la posición calculada con la odometría interna del robot (una vez que se pide al robot realizar cierto avance y/o giro, el robot reporta la distancia y ángulo que en realidad avanzó o giró, utilizando sus contadores de vueltas o *encoders*), para llevar cada medición del sonar a un punto (x,y) dentro del plano xy, ya que al conocer la ubicación y orientación del robot en cada paso, fijada inicialmente en (0,0) con una orientación de 0 grados, fue posible establecer un punto en el plano por cada medición del sonar, utilizando las siguientes ecuaciones:

$$Xp = Xr + d_i \cos(\theta + \phi_i) \tag{5.9}$$

$$Yp = Yr + d_i \sin(\theta + \phi_i) \tag{5.10}$$

(Xp,Yp) es la posición del punto en el plano xy, (Xr,Yr) es la posición actual del robot dentro del mismo plano, d_i es la medición del sonar i, θ es la orientación del robot respecto al marco de referencia y ϕ_i es la orientación del sonar i con respecto al frente del robot.

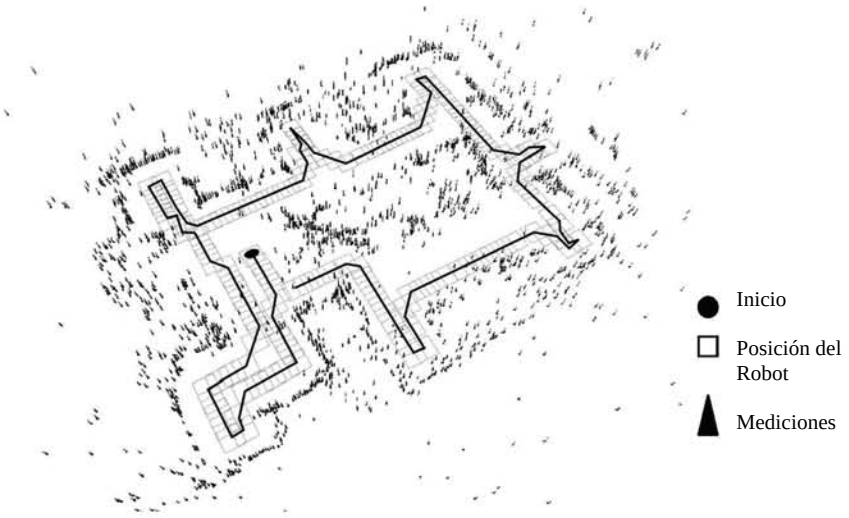

Figura 6.5: Mediciones con los sonares del T×8 en el ambiente real (llevada a puntos *x*,*y*) para el recorrido guiado.

Como resultado se genera un mapa (Fig. 6.5) con puntos (*x*,*y*) resultado de todas las observaciones realizadas. Como es de esperarse al tener errores en el movimiento del robot y errores en las mediciones de los sonares, dicho mapa no será demasiado realista y habrá que aplicar algunos métodos para filtrar el ruido (mediciones erróneas).

6.2.1 Diseño de la Red de Agrupamiento o Clustering

Como se mencionó en el capítulo 3, existe un tipo especial de red neuronal que puede hacer agrupamiento o "clustering", es decir, permite agrupar nubes de puntos próximos, dentro de una misma zona o *cluster*. Se pretende utilizar dicho tipo de red para identificar los objetos en el ambiente de trabajo.

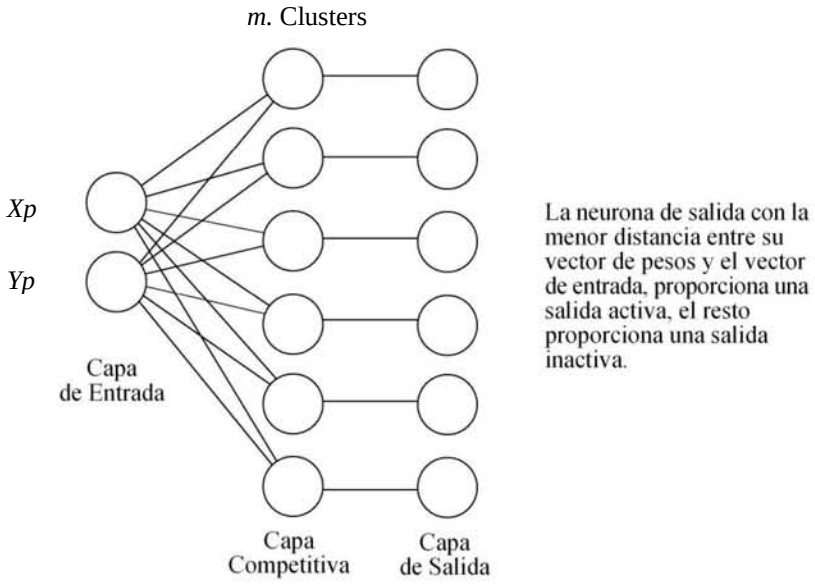

Figura 6.6: Red de Agrupamiento o *Clustering*.

Este tipo de red evalúa la distancia euclidiana entre todos los puntos (*x*,*y*) del mapa (correspondientes al mapeo de las lecturas de cada sonar al plano *x*,*y*) y cada uno de los vectores de pesos de las neuronas utilizadas (considerando cada vector de pesos como un punto *x*,*y*). Aquella neurona con la menor distancia al punto evaluado actualiza el valor de sus pesos con la ecuación (3.24), acercándolos en una fracción (factor de aprendizaje) hacia el punto evaluado. Como resultado del proceso anterior, luego de varias iteraciones las neuronas se han posicionado sobre los centroides de las nubes de puntos o *clusters*.

6.2.1 Prueba de la Red de Agrupamiento

Se implementó directamente la red neuronal en lenguaje C++ para poder utilizarla directamente en las rutinas de control del robot. En cuanto al número *m.* inicial de neuronas (que debe ser definido *a priori*) en este caso se decidió crear una malla que ocupara todo el espacio del ambiente de trabajo, ya que se conoce de antemano las dimensiones del mismo, sin embargo para un ambiente de dimensiones desconocidos se presentará el problema de no saber el número idóneo de neuronas que se deben utilizar. Se fijó un ritmo de aprendizaje γ=0.1 y se detuvo el entrenamiento cuando la diferencia entre los valores sucesivos de los vectores de pesos era menor a un 0.01% para todas las neuronas de la red.

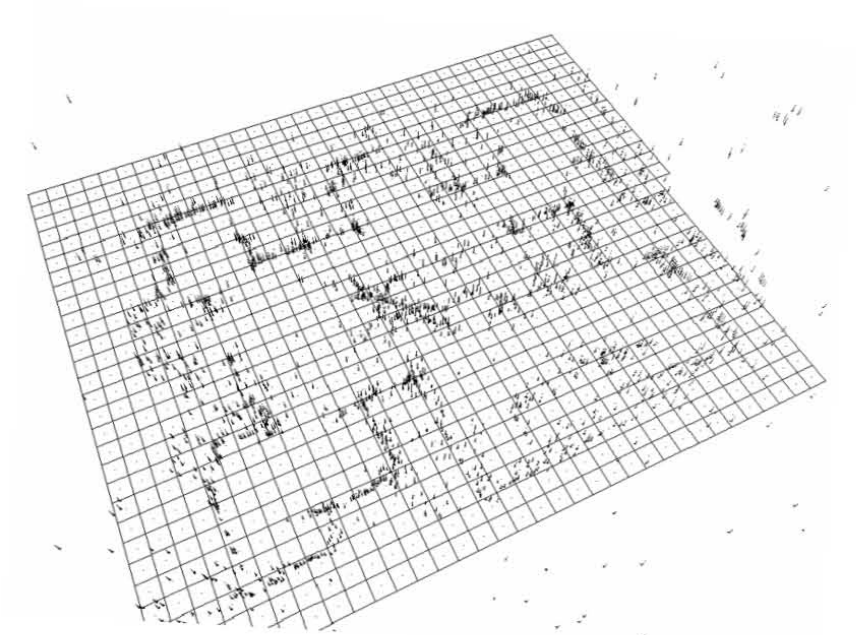

Figura 6.7: Colocación inicial de la malla de neuronas.

Una vez inicializados los pesos de la red, luego de aproximadamente 100 épocas de entrenamiento se observa que los vectores de pesos de algunas neuronas se han posicionado sobre las nubes de puntos que se presumen como objetos.

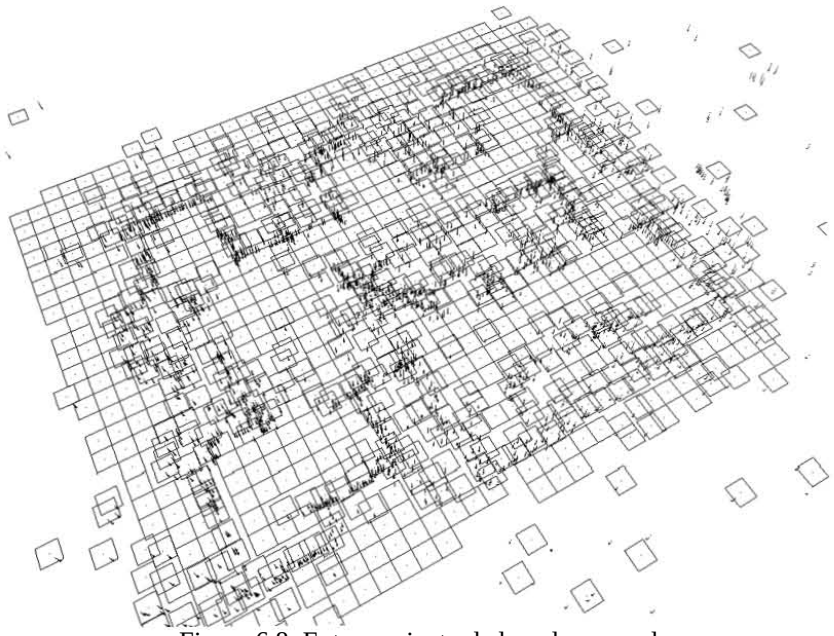

Figura 6.8: Entrenamiento de la red neuronal.

En este punto es posible asumir que aquellas neuronas cuyos pesos no fueron modificados, es decir, que no resultaron ganadoras en ninguna ocasión, se encuentran demasiado alejadas de las nubes de puntos y, por consiguiente, pueden ser eliminadas. Utilizando el procedimiento anterior permanecen exclusivamente aquellos grupos o *clusters* que corresponden a los objetos en el ambiente de trabajo (Fig. 6.9).

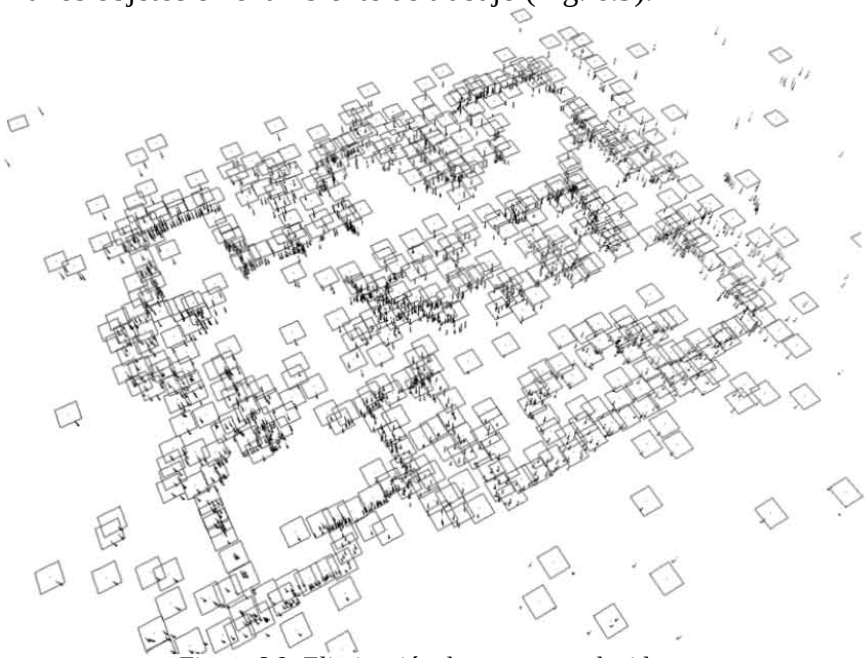

Figura 6.9: Eliminación de grupos no elegidos.

Finalmente cada objeto dentro del ambiente real de trabajo quedará representado por uno o varios grupos o clusters. De esta forma es posible asignar dimensiones espaciales a cada grupo (conforme a las dimensiones de cada celda en la malla original), con lo que se obtiene una representación espacial del ambiente de trabajo.

Figura 6.10: Representación espacial de los objetos del ambiente real.

Como se aprecia en la figura anterior, si son tomados como validos todos los puntos obtenidos a partir del mapeo al plano *xy*, al presentarse errores en las mediciones de los sonares (producto de un ángulo de incidencia oblicuo sobre el objeto) casi todo el espacio de trabajo se encontrará ocupado por objetos. Por este motivo hubo que desarrollar una estrategia para filtrar aquellos puntos producto de mediciones erróneas de los sonares.

6.3 Red Borrosa de Agrupamiento

Como se explicó anteriormente las redes de agrupamiento determinan la neurona más próxima a un punto en particular y acercan dicha neurona a ese punto. Sin embargo, el procedimiento anterior considera a todos los puntos por igual, sin importar si se trata de una lectura errónea o no.

Como se ha explicado anteriormente, al tomar en cuenta todos los puntos por igual se obtiene un espacio casi completamente saturado de objetos. Por el contrario, si se logra hacer que la red de agrupamiento opere de acuerdo a cierto grado de "certeza" o "confiabilidad" del punto evaluado y acerque la neurona hacia dicho punto, no sólo en función de un ritmo de aprendizaje previamente establecido, sino de la "confiabilidad" misma del punto en cuestión (ritmo de aprendizaje variable o dependiente) se podrá hacer que la red resulte menos sensible a entradas o mediciones erróneas y obtenga una representación espacial mucho mas acorde a la realidad.

6.3.1 Introducción

Las mediciones de los sonares son llevadas en forma de puntos al plano *xy* mediante las ecuaciones (5.9) y (5.10). Sin embargo, debido al error de medición que se presenta cuando el ángulo de incidencia de la señal del sonar no es perpendicular al contorno de los objetos, si se toman como válidos todos los puntos generados y se alimentan a la red de agrupamiento, las mediciones erróneas generan grupos o clusters correspondientes a objetos inexistentes en el ambiente real.

Por tal motivo es necesario eliminar las mediciones erróneas. Sin embargo, como el grado de error en la medición está íntimamente relacionado con el ángulo de incidencia de la señal del sonar sobre el objeto medido, resulta imposible saber al mismo tiempo de efectuar la medición con un sonar en particular, en qué ángulo incide la señal sobre el objeto medido y con ello el grado de error de dicha medición.

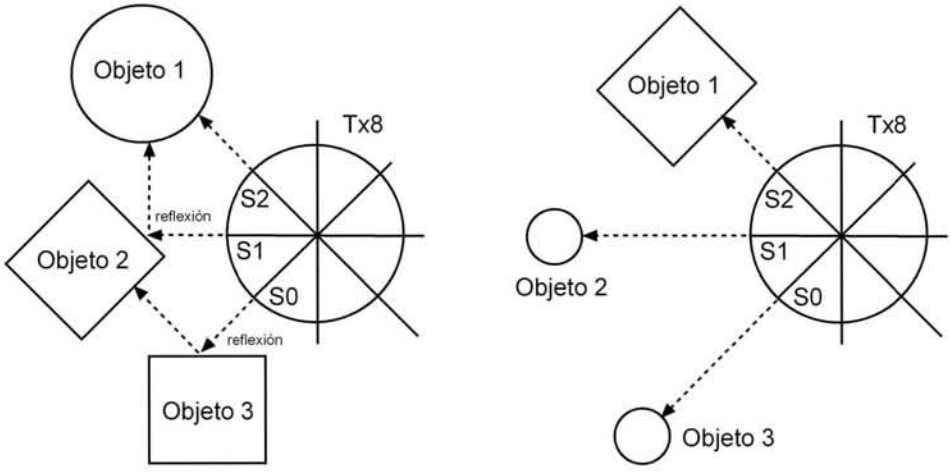

Figura 6.11: Diferentes objetos que dan lugar al mismo conjunto de mediciones.

6.3.2 Filtrado de las Mediciones Erróneas

Resulta imposible saber *a priori* cuáles mediciones son correctas y cuales no. Por este motivo, el primer criterio utilizado consistió en simplemente eliminar todos aquellos puntos del mapa, resultado de una medición d_i del sonar por encima de cierto valor o umbral (experimentalmente se determinó un umbral de 1.5 metros). Dicho umbral dependerá de las características particulares de cada ambiente y de la separación del robot con respecto a los objetos durante su recorrido (Fig. 6.12).

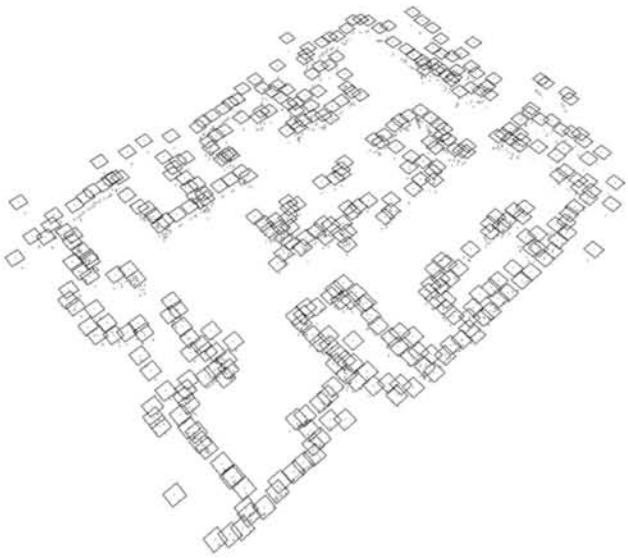

Figura 6.12: Filtrado de los puntos originados por mediciones mayores a 1.5 m.

Al eliminar los puntos, resultado de una medición por encima del umbral, se eliminan tanto puntos válidos como no-válidos y esto no eliminará medidas erróneas por debajo de dicho umbral, creando falsos objetos en el ambiente real. Por otro lado si se fija éste demasiado bajo, se corre el riesgo de eliminar muchas mediciones y generar un mapa vacío. Lo anterior significa que dicho umbral no podrá ser determinado *a priori* y los resultados dependerán de cada ambiente en particular.

6.3.3 Aplicación de Conceptos de Lógica Borrosa

Las redes autoasociativas como las *Redes de Agrupamiento* [Krose 96, pp. 57], *Mapas de Kohonen* [Kohonen 89], *Funciones de Base Radial* (RBFs) [Poggio 90] o las *Redes de Cuantificación Vectorial (VQ)*[Brío 02, pp.106], operan de una manera similar utilizando una serie de vectores como entrada. Sin embargo, en todos los casos no se cuestiona la validez de dichos vectores y son tomados por igual para los propósitos de cada red en particular.

Existe trabajo relacionado con el concepto de *Agrupamiento Borroso* (*Fuzzy Clustering*) [Höppner 99]. Sin embargo, éste se funda en la idea de asociar un nivel de pertenencia de cada vector, con respecto a cada cluster o grupo, de tal suerte que si el punto se encuentra al centro de un grupo, éste posea una alta pertenencia al mismo, mientras que si se encuentra en la periferia del mismo o entre dos clusters, éste posea un grado de pertenencia a ambos clusters por igual, sin embargo, de nueva cuenta no se cuestiona la validez de los vectores a agrupar (Fig. 6.13).

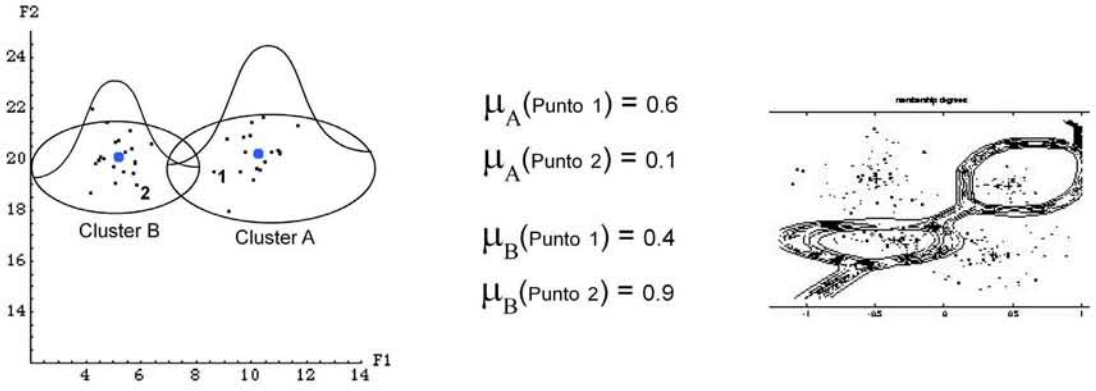

Figura 6.13: Agrupamiento Borroso (*Fuzzy Clustering*).

En este punto es posible hacer referencia a la teoría de conjuntos borrosos.

Sean dos conjuntos clásicos, el primero correspondiente a las edad de las personas consideradas como Adulto y el segundo el conjunto de mediciones que se tomadas como válidas para el proceso de filtrado de la sección anterior. Es posible tomar como edad adulta de los 18 a los 59 años de edad, mientras que como mediciones válidas de los sonares desde 10 cm. (mínima distancia posible reportada por los sonares) hasta 1.5 metros. Como conjuntos clásicos se indica con un *uno* si el valor pertenece al conjunto y *cero* si no pertenece (Fig. 6.14).

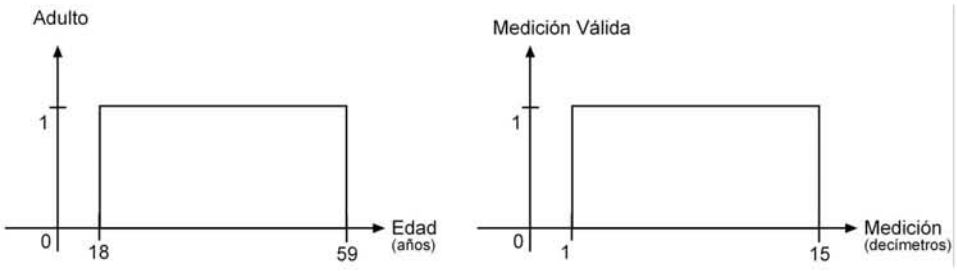

Figura 6.14: Conjuntos clásicos para "Adulto" (izquierda) y "Medición válida" (derecha).

El concepto de *conjunto borroso* permite contemplar un *nivel de pertenencia* al conjunto, distinto de cero o uno. Ahora es posible definir un valor de pertenencia *cero* para la edad de 16 años e incrementarlo de manera uniforme hasta alcanzar los 21 años, donde se mantiene en el máximo para comenzar a descender a partir de los 55 años, hasta llegar nuevamente a *cero* a los 65 años. De la misma forma es posible considerar que cuanto más pequeña sea una medición otorgada por el sonar, mayores posibilidades tendrá de ser una medición correcta (inclusive para los casos en que el sonar presenta lecturas sucesivas diferentes, estando en la misma posición y orientación). Con esto es posible fijar una pertenencia de *uno* a las mediciones desde un valor d=0 (imposible de obtener con los sonares en la práctica), y hasta d=1 (si bien los sonares no entregan lecturas por debajo de 10 cm. no existe razón para considerarlas inválidas). El valor de pertenencia desciende progresivamente hasta alcanzar el umbral impuesto de 1.5 metros con un valor de pertenencia igual a *cero* (Fig. 6.15).

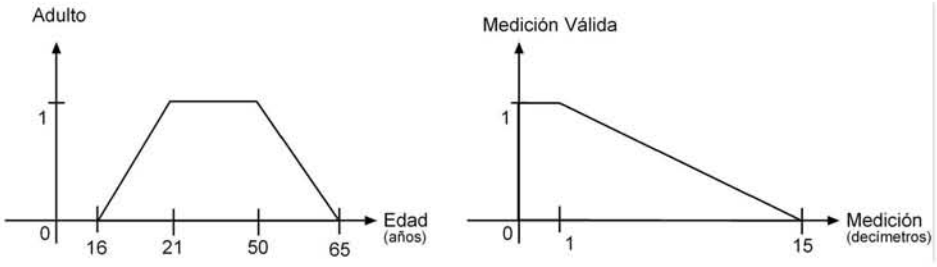

Figura 6.15: Conjuntos borrosos para "Adulto" (izquierda) y "Medición válida" (derecha).

Si a cada valor de distancia d_i medido por el sonar, se le asigna un nivel de pertenencia al conjunto de mediciones válidas, de acuerdo con la función de pertenencia antes descrita y si se considera por simplicidad que se conoce perfectamente la ubicación y orientación del robot al momento de tomar las lecturas con el sonar, al aplicar las ecuaciones (5.9) y (5.10) para generar un punto en el plano *xy*, tanto la posición (Xr,Yr) como la orientación θ serán valores no borrosos y con ello $\cos(\theta+\phi_i)$ y $\sin(\theta+\phi_i)$. Al aplicar el principio de extensión (ver cap. 4.5) se tiene que:

$$\begin{aligned}\mu(Xp) &= \sup\{\min[1,\sup\{\min[\mu(d_i),1]\}\ ]\}\\ &= \sup\{\min[1,\mu(d_i)]\}\\ &= \mu(d_i)\end{aligned} \qquad (6.1)$$

De forma similar se obtiene

$$\mu(Yp) = \mu(d_i) \tag{6.2}$$

Como los valores $\mu(Xp)$ y $\mu(Yp)$ siempre serán idénticos, al momento de llevar al plano x*y* el punto (Xp, Yp) se puede tomar

$$\mu(Xp, Yp) = \min[\mu(Xp), \mu(Yp)] = \mu(d_i) \tag{6.3}$$

En la práctica, conforme el robot se mueve dentro del ambiente, se pierde certeza en la posición y orientación, de tal suerte que en las ecuaciones (5.10) y (6.1) todos los términos involucrados deberán ser tomados como borrosos, modificando con ello $\mu(Xp, Yp)$.

6.3.4 Determinación de la Función de Pertenencia Óptima

Analizando la Figura 6.15 se observa que, no obstante el valor de pertenencia desciende de manera constante hasta el umbral fijado en 1.5 metros, se descartan por completo las mediciones por encima de dicho valor. Supóngase el caso en el que se tiene un sólo punto generado a partir de una lectura por debajo del umbral y 10^6 puntos generados con lecturas mayores al mismo, todos en la misma región y a corta distancia del primer punto (Fig. 6.16).

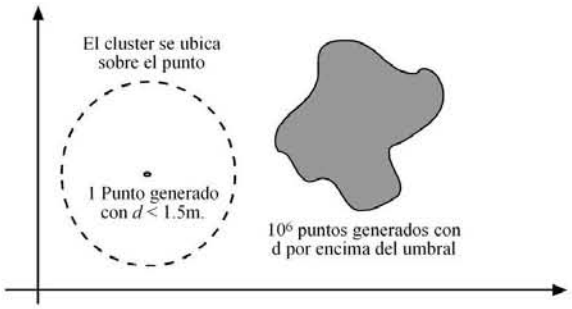

Figura 6.16: El cluster se centra exclusivamente en los puntos generados con $d < 1.5$ m.

Evidentemente el cluster quedará ubicado sobre la única lectura, sin embargo sería deseable que aquellas mediciones lejanas, posean un nivel de "confiabilidad" muy pequeño, pero no nulo, ya que el seguimiento de paredes utilizado por el robot, no garantiza encontrar los objetos al centro de una habitación grande, si la distancia del robot a los mismos supera el umbral, con lo que el mapa quedaría incompleto.

Por otro lado resulta completamente arbitrario considerar con menor confiabilidad las mediciones realizadas a 75 cm., contra aquellas realizadas a 30 cm., ya que pueden existir mediciones perfectamente válidas a 75 cm. Sería conveniente considerar aproximadamente con similar confiabilidad mediciones en cierto intervalo, descender rápidamente en cuanto la distancia se acerque a ciertos límites y continuar asintóticamente muy cerca del valor *cero*, conforme la distancia de la medición se incremente.

Una función que cumple con los requisitos anteriores es la función gaussiana:

$$f(x)=ke^{\frac{-x^2}{2\sigma^2}} \tag{6.4}$$

ya que presenta su valor máximo para x=0, comienza a descender suavemente en un pequeño intervalo y lo hace rápidamente al acercarse a σ. Su valor es prácticamente cero por encima de 3σ y continúa asistóticamente al infinito. Por estas razones se decidió utilizar una función gaussiana como función de pertenencia.

Al elegir una gaussiana como función de pertenencia, mientras más grande sea el valor entregado por el sonar, menor confiabilidad tendrá y viceversa, de tal suerte que la mejor medición (valor de pertenencia =1.0) será aquella que sea realizada a una distancia cero del objeto, sin embargo se sabe que esto no es posible, por una parte debido a que no se puede permitir que el robot colisione con los objetos, y por otra debido a la zona "ciega" de los sonares que entrega una medición mínima de 10 cm.

Al analizar los datos obtenidos durante el recorrido efectuado por el robot, se observó que no se alejó por encima de 80 cm. del contorno de los objetos y en promedio se mantuvo a una distancia de 50 cm. Con base en esta información se determinó una distancia máxima de 150 cm. para ser tomada en cuenta. Lo anterior significa en términos prácticos que se tomarán en cuenta únicamente las mediciones hechas a una corta distancia del robot (en su mayoría las mediciones correspondientes a los objetos o paredes cuyos contornos el robot está siguiendo).

Conforme a esta información se determinaron los parámetros de la gaussiana *k=1* y σ=5 en la ecuación (6.4), resultando la siguiente expresión que expresa la *confiabilidad* en función de la distancia en decímetros:

$$\mu(d)=e^{\frac{-d^2}{50}} \tag{6.5}$$

Con estos parámetros, a una distancia *d* de 80 cm. (expresada como 8 dm.) la función proporciona una confiabilidad de 0.27, mientras que para una distancia de 30 cm. la confiabilidad es 0.83 y para 150 cm. es de sólo 0.01. Por encima de 150 cm. la función retorna una confiabilidad prácticamente igual a *cero*, pero no *cero*.

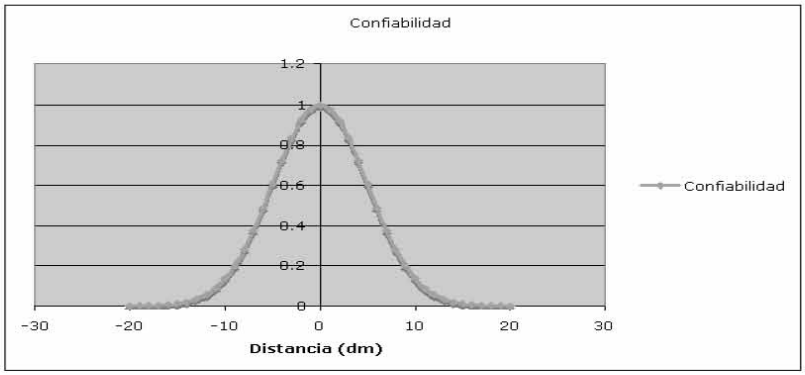

Figura 6.17: Función de pertenencia de la *confiabilidad* en la medición.

Cabe aclarar que, aunque parezcan un poco rígidos los parámetros de la función de pertenencia y ésta deje fuera información que en principio puede ser correcta, permite eliminar directamente los errores resultantes de las reflexiones, por rebasar la desviación máxima a una distancia de 50 cm. (distancia máxima a la cual se alejó el robot de los objetos) que, según se puede apreciar en la sección 5.10.2 puede comenzar a dar valores erróneos desde 90 cm. (μ=0.19) para metal, hasta 120 cm. (μ=0.06) y 170 cm. (μ=0.003) para cartón y tela respectivamente.

Si se grafica nuevamente cada punto obtenido a partir de (5.9), (5.10) y (6.1) como un triángulo en la posición (Xp, Yp), cuya altura esté en función del valor de pertenencia o *confiabilidad* de dicho punto, se obtiene la siguiente figura:

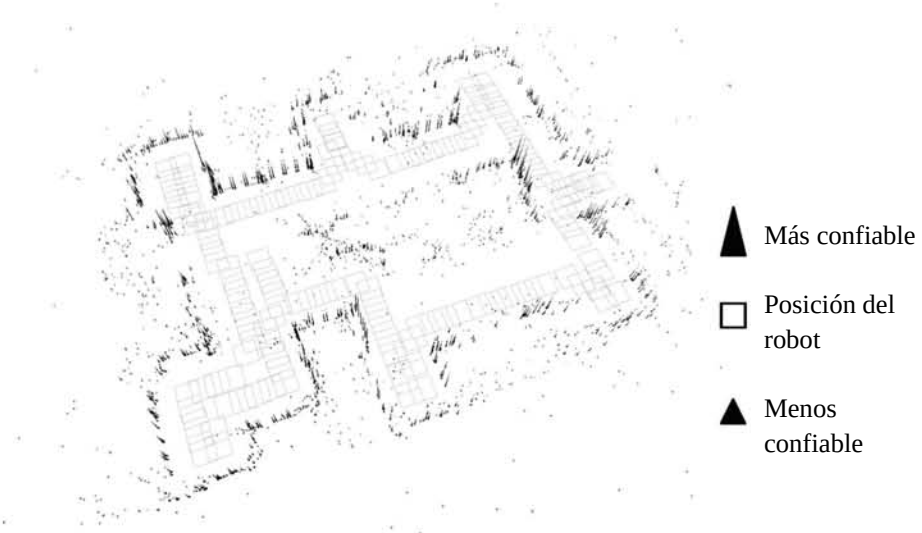

Figura 6.18: Puntos obtenidos en el ambiente real. La altura indica el grado de confiabilidad.

Al ampliar una sección se observa que aquellas lecturas próximas al robot resultan más confiables que aquellas que fueron realizadas a una distancia considerable. De igual forma puede apreciarse que la mayoría de las lecturas erróneas que obstaculizan el paso del robot poseen un valor de confiabilidad muy bajo o prácticamente nulo.

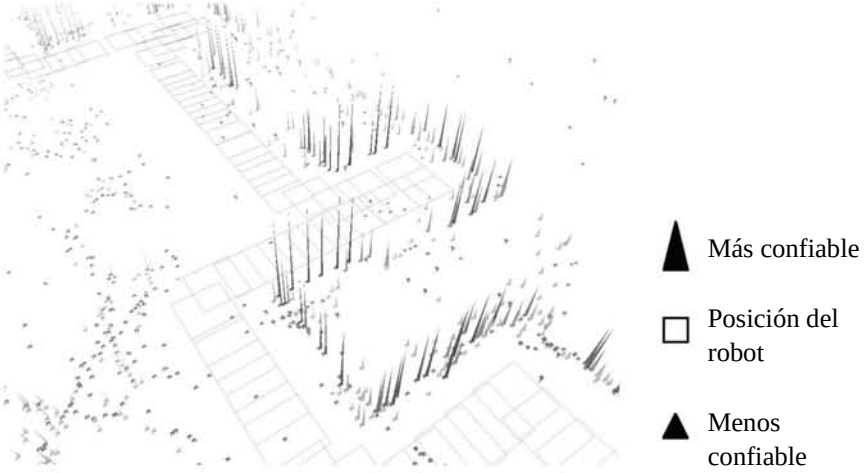

Figura 6.19: Confiabilidad de las lecturas y camino recorrido por el robot.

6.3.5 Adición de Valores Borrosos a la Red de Agrupamiento

Entonces surge el problema de modificar la *Red de Agrupamient*o para aceptar como entrada un conjunto de vectores con cierto valor de pertenencia y proporcionar como salida un conjunto no-borroso de grupos o clusters que representen los objetos del mapa métrico del ambiente (Fig. 6.20).

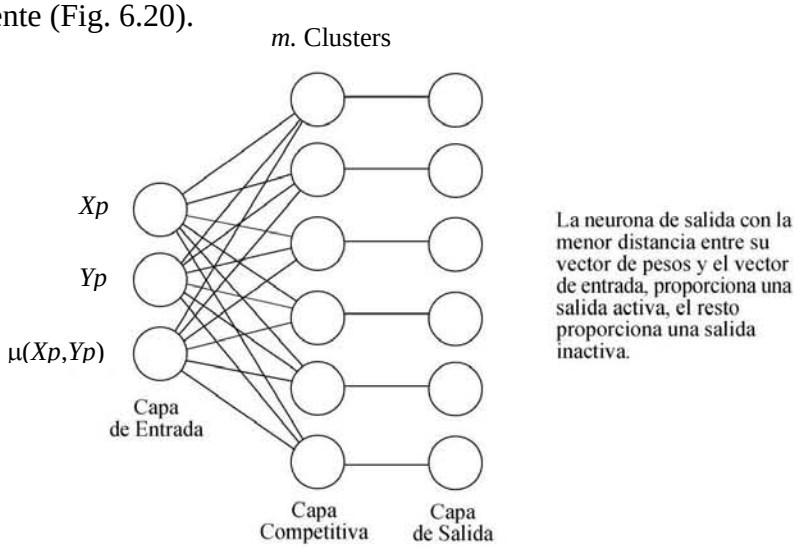

Figura 6.20: Red Borrosa de Agrupamiento.

La salida debe ser no-borrosa para poder determinar con precisión la ubicación de los objetos. Nada impide trabajar con mapas con contornos borrosos, sin embargo, esto implica utilizar redes de conectividad borrosas, rutas borrosas, búsquedas borrosas, etc., lo cual complicaría el proceso de cálculo y navegación.

La regla de actualización de la neurona ganadora par las Redes de Agrupamiento es:

$$\overline{W}_{i+1}=\overline{W}_i+\gamma(\overline{X}_i-\overline{W}_i) \qquad (6.6)$$

(donde $\overline{W}_{i+1}$ es el vector de pesos de la neurona ganadora al tiempo i+1, $\overline{W}_i$ es el vector de pesos de la misma neurona al tiempo i, $\overline{X}$ es el vector alimentado a la red al tiempo i y γ es el ritmo de aprendizaje), la neurona ganadora acerca su vector de pesos en una fracción γ del vector que lo lleva directamente hacia el punto $\overline{X}$ (Fig. 6.21).

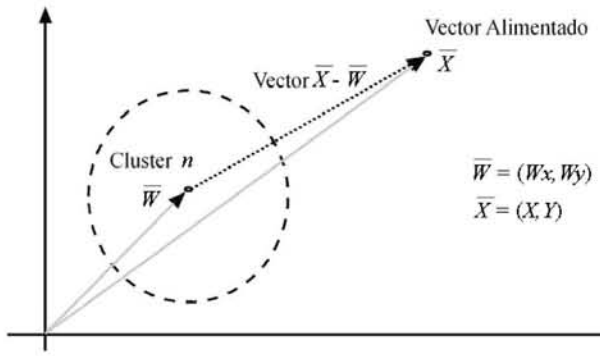

Figura 6.21: Proceso de aprendizaje en las redes de agrupamiento.

Ahora se desea que la red acerque el vector $\overline{W}$ de la neurona ganadora hacia el punto $\overline{X}$ en función del grado de pertenencia de dicho punto al conjunto de mediciones que se consideran como "confiables". Si el punto es "confiable" la red podrá acercar el vector $\overline{W}$ hacia $\overline{X}$ con el factor de aprendizaje γ previamente fijado. Si el punto no es confiable en absoluto la red no deberá modificar el vector $\overline{W}$ de la ganadora.

El nivel de pertenencia del punto $\mu(\overline{X})$ va desde 0 hasta 1. Si se toma directamente dicho valor como un factor de ajuste del ritmo γ de aprendizaje, entonces se tiene

$$\overline{W}_{i+1} = \overline{W}_i + \gamma\left[\mu(\overline{X})\right](\overline{X}_i - \overline{W}_i) \tag{6.7}$$

con lo cual se logra que la red proporcione clusters no-borrosos para una entrada de vectores con cierto grado de "confiabilidad" (Fig. 6.22).

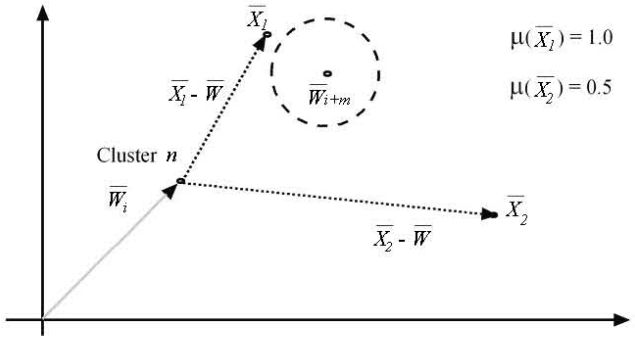

Figura 6.22: Proceso de aprendizaje en la red borrosa de agrupamiento.
El cluster se acerca al punto con el mayor nivel de *confiabilidad.*

6.3.6 Adición de Otros Factores como el Factor Tiempo

El modelo anterior permite agregar otros factores a la regla de aprendizaje de la red, por ejemplo, un factor en función del tiempo *i* en el cual se realizó la medición. En la práctica el robot comete errores de movimiento por lo que, luego de cierto tiempo, su posición y orientación serán completamente distintas a aquellas calculadas exclusivamente mediante la odometría interna del robot y tomadas como base para el cálculo de los puntos en el plano. Mientras más tiempo pase, mayor será el error de localización y al momento de utilizar la red borrosa de agrupamiento se obtendrán clusters en ubicaciones erróneas.

Si se define cierto intervalo de tiempo a partir de la última lectura y se considera que durante dicho intervalo no se ha cometido un error de movimiento grande, no obstante la localización y orientación actual del robot sean erróneas, el mapa generado durante dicho intervalo tendrá una correcta distribución espacial de los objetos. Únicamente se encontrará girado y desplazado de su posición real. Si se generan sucesivamente mapas locales a intervalos regulares éstos pueden ser unidos en un solo mapa mediante técnicas como las referidas en [Amigoni 03].

De esta forma es posible definir otro conjunto borroso cuyo valor de pertenencia esté en función de que tan "reciente" es la medición, con respecto al instante en el cual se está efectuando la actualización de la neurona ganadora (Fig. 6.23).

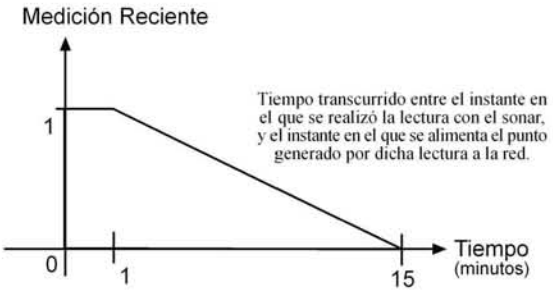

Figura 6.23: Otro posible factor borroso.

Lo anterior permitiría a la red ignorar lecturas lejanas en el tiempo, con lo que se generaría un mapa "local" del ambiente visitado recientemente por el robot, con una mejor localización espacial de los objetos, aunque girada y desplazada de su posición y orientación reales.

Si se agrega este factor a la nueva regla de actualización, se tiene

$$\overline{W}_{i+1} = \overline{W}_i + \gamma\left[\mu(\overline{X})\right]\left[\mu(_{i-m})\right](\overline{X}_i - \overline{W}_i) \qquad (6.8)$$

donde *i* es el instante actual y *m.* el instante donde se realizó la lectura que dio origen al vector $\overline{X}$, alimentado en el instante *i*.

De esta forma es posible agregar sucesivamente tantos factores como se desee, tales como el grado de certeza en la ubicación u orientación al momento de efectuar la medición, el grado de certeza en la ubicación al momento de actualizar la neurona ganadora, etc., sin embargo es posible multiplicar todos esos factores entre sí agrupándolos en un sólo factor que modifique el ritmo de aprendizaje γ. Es posible combinar todos los valores de pertenencia, no sólo mediante una multiplicación simple, sino mediante alguna operación o combinación de operaciones borrosas (como las especificadas en la sección 4.3.2) y determinar una sola $\mu(\overline{X})$ en función de todos los valores de pertenencia asociados a la misma. Esto supone utilizar la ecuación (6.7) que posee un solo factor borroso, sin embargo en cada iteración es necesario calcular $\mu(\overline{X}) \ \ \forall \ \overline{X}_{i,m}$ previo a la alimentación del vector a la red de agrupamiento.

6.3.7 Operación de la Red Borrosa

Mediante el método anterior se etiquetan nuevamente todos los puntos (*x*,*y*) encontrados con el valor de su confiabilidad según la magnitud de la medición que les dio origen (no se tomó en cuenta el factor tiempo) y se entrena nuevamente la red de agrupamiento (se utilizaron las mismas 100 iteraciones que la red original).

En este caso, debido a que no se filtran las mediciones erróneas, para todos los puntos la capa competitiva de la red encuentra el cluster más próximo a cada vector de entrada, y acerca su vector de pesos hacia dicho punto, dependiendo del valor de pertenencia del mismo.

Para eliminar los clusters del espacio libre, no bastará con determinar cuáles no fueron elegidos como los más cercanos a algún punto (Fig. 6.9). Como cada punto en el plano modificará en mayor o menor medida la ubicación del cluster próximo a él, los puntos con un valor de pertenencia pequeño, prácticamente no modificarán la posición del cluster próximo a ellos, pero habrá de cualquier forma una modificación.

Para determinar cuáles clusters son eliminados al no considerar objetos en ellos, se optó por establecer un umbral. En cada iteración, una vez que se han alimentado todos los puntos a la red, por cada cluster se suma el nivel de pertenencia de todos los puntos que resultaron próximos a él y modificaron su posición. Por ejemplo, cierto cluster resultó cercano a tres puntos con niveles de pertenencia 0.1, 0.1 y 0.001, si se establece un umbral de 0.3 como mínimo, es claro que el cluster anterior será descartado como candidato de la lista final. Si en cada iteración se eliminan los clusters que no superen el umbral, en la siguiente iteración para el ejemplo anterior, la red deberá encontrar el nuevo cluster más cercano a los 3 puntos y modificará su valor.

Si existiese un número grande de puntos con muy poca confiabilidad y, luego de realizar la suma de sus valores de pertenencia en una iteración, resultase que dicha suma no supera el umbral impuesto para “crear” un nuevo cluster, en la iteración siguiente, dichos puntos podrán afectar la ubicación del cluster más cercano a ellos.

En la Figura 6.24 se muestra el resultado del proceso anterior, con un umbral de 0.1 para determinar los clusters definitivos.

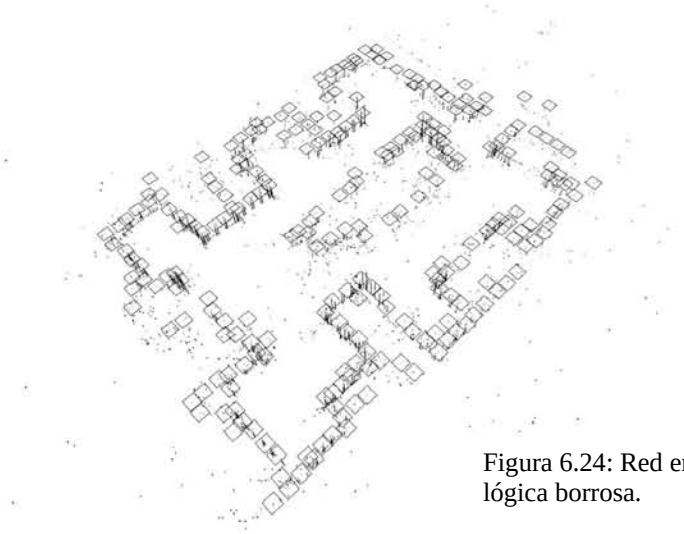

Figura 6.24: Red entrenada con lógica borrosa.

Como puede observarse, los resultados son mucho mejores, ahora ya aparecen zonas claramente libres. Para entender lo sucedido se muestra una sección donde se observa cómo aquellas mediciones más confiables acercan hacia ellas a las neuronas en mayor medida que los puntos poco confiables o de confiabilidad nula.

Figura 6.25: Ampliación de grupos encontrados para la misma zona del mapa: izquierda) empleando todos los puntos. centro) filtrado de lecturas mayores a 1.5 m y derecha) red borrosa de agrupamiento. Las zonas obscuras representan los contornos reales de los objetos.

Con la adición del factor borroso, la red de agrupamiento genera mejores contornos para los objetos que el método de discriminación mediante un umbral, ya que al irse alejando de los mismos, el error de movimiento del robot evita que perciba el contorno de los objetos en la misma posición. La red borrosa permite que los clusters correspondientes a los contornos de los objetos, se encuentren más próximos a las lecturas con mayor *confiabilidad* y evita que se desplacen demasiado, conforme el robot se aleja, al disminuir dicho grado de *confiabilidad*. Por otro lado, para la red borrosa de agrupamiento no basta que exista una punto en cierta posición para que genere un cluster en dicha posición. La red evalúa la confiabilidad de dicho punto, si es alta, crea un cluster en dicha posición, de lo contrario, busca el candidato más cercano y lo modifica. Lo anterior pone de manifiesto la mejora obtenida al aplicar los conceptos de lógica borrosa en la operación de las Redes de Agrupamiento.

Finalmente, para crear el mapa métrico se asume que se encuentra un objeto de las dimensiones definidas para cada cluster, en la posición (x,y) correspondiente al centroide del cluster generado.

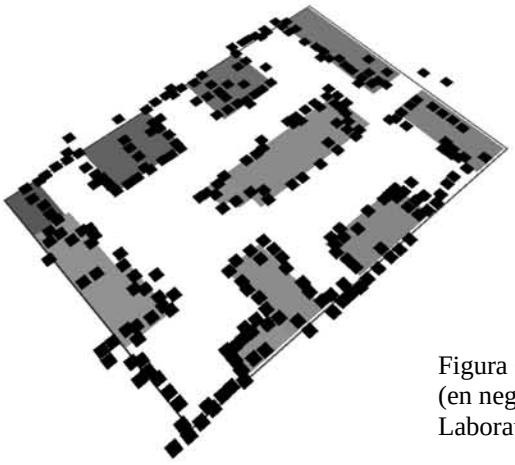

Figura 6.26: Mapa métrico elaborado (en negro) e ideal (en gris) del Laboratorio de Biorrobótica.

La ventaja que supone este método para crear el mapa métrico, es que puede realizarse en tiempo real mientras el robot navega, es decir, conforme se generan los nuevos puntos producto de las observaciones, dichos puntos entran a la lista de puntos con los que se esta entrenando la red de agrupamiento, de tal suerte que es posible aprovechar el tiempo entre movimientos sucesivos del robot para entrenarla.

Como puede observarse en la Figura 6.26, existen muchos espacios huecos sobre los contornos de los objetos, los cuales se deben a que, en su mayoría, se trata de escritorios o mesas cuya en cuya parte inferior se encuentran colocados objetos o sillas pero que no forman una superficie continua. Por otro lado al interior de los objetos no existen grupos ni clusters debido a que es muy raro que el sonar penetre en los objetos, salvo en el caso de que sean varios objetos de cartón o plástico, apilados debajo del escritorio o mesa. Se deberá tener cuidado para evitar que los algoritmos de movimiento y planeación de rutas traten de introducir al robot por tales aberturas.

6.3.8 Determinación del Número Óptimo de Grupos o Clusters

Existen diversas opciones para determinar el número apropiado de clusters. En [Kuri 02] se propone un método para determinar dicho número, esto debe hacerse fuera de línea, una vez obtenidas las mediciones del ambiente y probando con distinto número inicial de clusters. Debido a que se requiere hacer el mapa en línea, el método anterior no resulta de mucha utilidad. Se optó por utilizar un método de creación de clusters sobre demanda que permitiese crear nuevas neuronas dinámicamente mientras el robot navega. El algoritmo opera de la siguiente forma:

1. Se inicia con un sólo cluster.
2. Para cada punto (x,y) se determina el cluster (neurona) más cercano.
3. Si la distancia a dicho cluster excede un valor δ predeterminado y la confiabilidad del punto supera un umbral α
   3.1. Se crea un nuevo cluster (neurona) situado sobre el punto (x,y)

de lo contrario

   3.2. Se acerca el centroide del cluster ganador hacia el punto en cuestión mediante la ecuación (6.7).

Para el presente trabajo la distancia máxima δ es sinónimo del radio de influencia de cada cluster (se definió como el doble del radio del robot para todos los clusters) mientras que $\alpha = 0.1$. Con el método anterior aseguramos que cada punto esté cerca de un cluster. Como resultado en sólo cinco épocas de entrenamiento se ha alcanzado el mismo número de clusters que la red que inicia con un número predeterminado y luego elimina los no utilizados. En otras 10 épocas la red ha se entrenado perfectamente. Nuevamente se fijó el ritmo de aprendizaje γ=0.1 y se detuvo el entrenamiento cuando la diferencia entre todos los valores sucesivos de los vectores de pesos era menor a un 0.01%.

Figura 6.27: Mapa métrico con clusters sobre demanda.

Otra gran ventaja de utilizar clusters sobre demanda es la rapidez con la cual se entrena la red, ya que si se define inicialmente un número grande de neuronas cuando no usamos este método, es necesario evaluar para cada punto la distancia contra todas las neuronas de la red en cada iteración, lo que hace lento el aprendizaje. Por el contrario cuando se utiliza el agrupamiento sobre demanda, en las primeras iteraciones se crean los clusters necesarios y en las siguientes la red se dedica únicamente a moverlos de posición, además, coloca nuevos clusters en las zonas donde existen puntos, mientras que en el otro caso si no se conocen las dimensiones para crear la malla o los clusters de inicio se definen al azar, en muchas ocasiones pueden quedar demasiado lejos de todos los puntos y nunca ser tomados en cuenta, mientras que zonas densamente ocupadas pueden quedar representadas con un solo cluster si una sola neurona se encuentra próxima a esa ubicación.

6.4 Creación del Mapa Topológico

Existen diferentes métodos para crear el mapa topológico (como los referidos en el capítulo 2), en este caso se utilizó una malla rectangular con celdas cuadradas para ocupar todo el espacio sobre el mapa métrico, lo anterior implica simplificar notablemente el cálculo de la posición de los nodos navegables. Se fijó el ancho y largo de cada celda igual que en el espacio simulado (20 cm.).

Una vez que se han encontrado los objetos en el mapa métrico se determinan las celdas navegables, para tal efecto se evalúa si el robot (con cierta distancia de seguridad) puede situarse perfectamente en el centroide de cada celda.

Evidentemente las celdas libres serán aquellas que no se encuentren ocupadas por ningún cluster o que, si bien el cluster se encuentra parcialmente sobre la celda, permita al robot colocarse (con todo y distancia de seguridad) sobre el centroide de dicha celda.

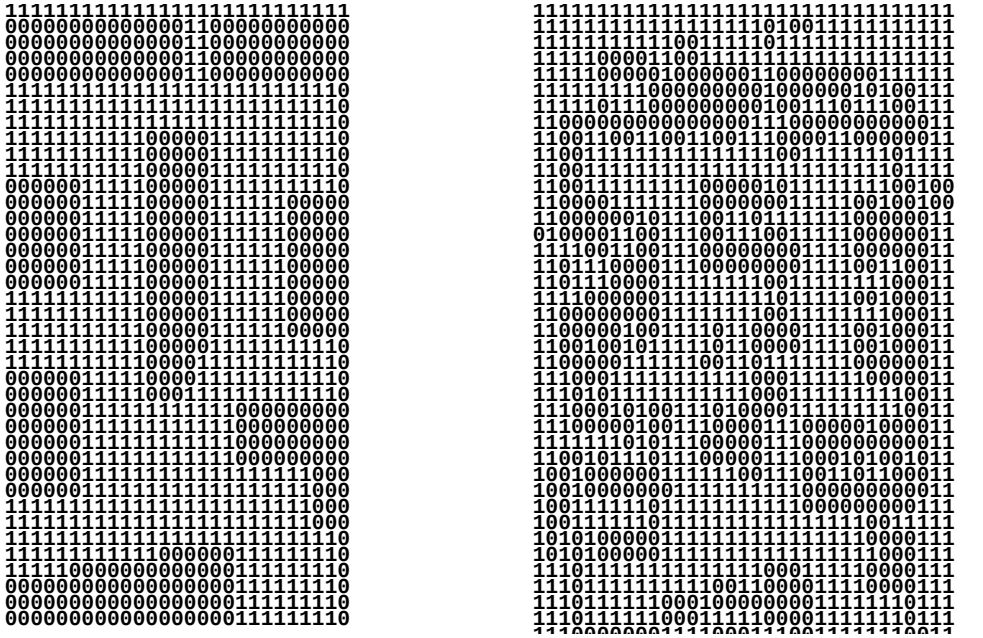

Figura 6.28: Mapa ideal (izquierda) y mapa elaborado con la red borrosa de agrupamiento (derecha).

En cuanto a las dimensiones totales del mapa se toman las dimensiones del rectángulo más pequeña que contenga todos los grupos o clusters, es decir, la *x* mínima y máxima entre todos los clusters, y la *y* mínima y máxima de entre todos ellos. Lo anterior garantiza que se contemple dentro del mapa toda el área donde se encuentran ubicados los objetos, obviamente en muchos casos mayor que la superficie navegable por el robot.

Como puede verse, se obtuvieron mejores resultados utilizando la red de agrupamiento borrosa que con el algoritmo simple de bloqueo de puntos (Fig. 6.4) ya que las zonas navegables son más amplias y, no obstante que el mapa resultante es de dimensiones mayores, se apega más a la realidad. Por otro lado, ya no se observan los objetos "fantasma" bloqueando el paso del robot y si bien existen muchas celdas libres al interior de los objetos, éstas pueden ser ignoradas si se elimina la conectividad a tales nodos.

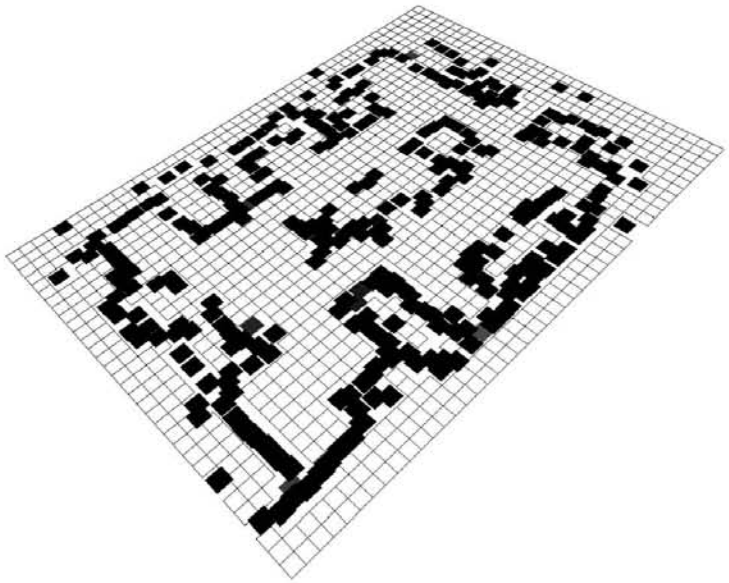

Figura 6.29: Mapa métrico y nodos libres.

6.4.1 Creación de la Matriz de Conectividad

Una vez que se ha determinado la ubicación de cada nodo y si se encuentra libre u ocupado por algún objeto, es necesario verificar la conectividad o navegabilidad entre todos los nodos, es decir, si el robot puede navegar desde un nodo origen a un nodo destino, para todos los nodos libres del mapa.

Para tal efecto se debe calcular por cada par de nodos si le es posible al robot trasladarse libremente y en línea recta entre dichos nodos (sin colisionar con ningún objeto). Para esto es necesario definir una estrategia que permita determinar si se encuentra algún objeto interponiéndose en la trayectoria de movimiento o, de no ser así, si el robot logra pasar alejado de los objetos con cierto margen de seguridad. De ser posible la navegación entre estos dos nodos, se indica un uno en la celda (origen, destino) de una matriz cuadrada de dimensiones $n \times n$ (donde n es el número de nodos libres de la red), de no ser posible la navegación se indica un cero en esa posición.

En dicha matriz cada celda representa la posibilidad de ir desde el nodo *renglón* al nodo *columna*. De esta forma el valor (10,2) indica la posibilidad de ir del nodo libre 10 al nodo libre 2. Si el valor en dicha posición es cero será indicativo de que no existe conectividad entre dichos nodos. El resto de los nodos (los no navegables) no se consideran en la matriz de conectividad, ya que, de entrada, no pueden existir conexiones a los mismos e incluirlos en la matriz de conectividad resultaría costoso en términos de memoria, ya que, por ejemplo si se tiene un mapa de 30×30 nodos, se debería tener una matriz de conectividad de $900 \times 900 = 810{,}000$ casillas aunque sólo existiesen 100 nodos libres en el mapa, lo que implicaría una matriz de sólo $100 \times 100 = 10{,}000$ casillas. Otro dato importante es que la matriz es simétrica, es decir, la navegabilidad (i,j) es igual a la (j,i) ya que si es posible ir de un nodo i a un nodo j, obviamente deberá ser posible ir ahora del nodo j al nodo i. Por esta razón es posible disminuir el número de casillas de la matriz a la mitad, si se maneja dentro de un vector unidimensional, obteniéndose una función que indica la posición sobre el vector para un (i,j) dados.

Finalmente se determinó una conectividad igual a cero cuando $i = j$ para evitar que el algoritmo de búsqueda de rutas intente buscar conexiones entre el mismo nodo como origen y destino.

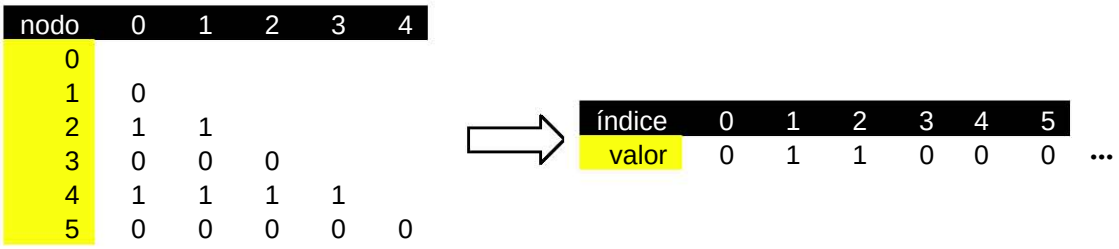

Figura 6.30: Transformación de una matriz de conectividad triangular en un vector unidimensional
Los números dentro de las casillas de la matriz corresponden al índice del vector.

```
1. Si i<j

      1.1 Indice = (j*(j-1))/2 + i

de lo contrario

      //como la matriz es simétrica
      1.2 Indice = (i*(i-1))/2 + j
```

Figura 6.31: Algoritmo para determinar el índice del vector unidimensional correspondiente a la conectividad entre los nodos i y j.

Con el método anterior es posible reducir la matriz de dimensiones $n \times n$ a un vector de dimensiones $n(n+1)/2$. Para el ejemplo citado es de $100 \times 101/2 = 5{,}050$ posiciones.

Otro método ampliamente utilizado consiste en crear una lista de aristas navegables, donde se indica por cada arista el nodo origen y el nodo destino. En el ejemplo, suponiendo

que existen 100 nodos libres y existiese conectividad entre los 100 nodos (en el caso de un área circular por ejemplo) se tendrían $100 \times 100 \times 2 = 20{,}000$ posiciones si se evalúan todas las posibilidades o $100 \times 101/2 \times 2 = 10{,}100$ posiciones si no importa el orden (origen, destino) en la lista de aristas.

Puede verse fácilmente que el método anterior resulta cómodo cuando se tiene un número limitado de aristas navegables ya que, el principal problema que plantea, consiste en la búsqueda de la conectividad entre dos nodos *i* y *j*, pues es necesario buscar en toda la lista de aristas, tanto en el nodo *origen* como *destino*, lo cual hace que el algoritmo de búsqueda de rutas se torne lento, por esta razón se decidió utilizar el vector de conectividad antes citado.

Por otro lado también es posible elaborar la matriz de conectividad evaluando exclusivamente la posibilidad de navegar a los nodos vecinos del nodo actual, lo cual si bien reduce notablemente el número de conexiones o aristas entre nodos, evitaría la posibilidad de encontrar una rápida solución para ir directamente (en un solo paso) de un nodo origen a un nodo destino ya que, aunque esto fuera posible, la ruta encontrada iría mostrando las celdas o nodos vecinos por los que hay que pasar hasta llegar al destino. Al hablar de nodos vecinos se hace referencia a las celdas contiguas en la malla definida como mapa topológico.

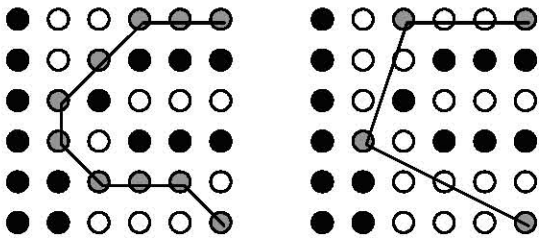

Figura 6.32: Rutas encontradas utilizando conectividad local (izquierda) y total (derecha).

Como se observa en la figura, el método implementado en el presente trabajo, que evalúa la conectividad entre todos los nodos del mapa topológico, permite encontrar rutas de distancia mínima y un número reducido de pasos (nodos visitados).

Finalmente cabe aclarar que, si en lugar de marcar la conectividad como 1 o 0 indicamos la distancia que existe entre el nodo *i* y el nodo *j*, automáticamente estamos asignando un "peso" a la conexión, lo cuál resulta de mucha utilidad para encontrar la ruta de menor distancia entre dos posiciones dentro del mapa métrico.

En cuanto al algoritmo para determinar si es posible navegar en línea recta desde un nodo origen a un destino existen varias posibilidades, una de ellas (la más simple) consiste en trazar una línea recta que una dichos nodos y determinar si intersecta alguna de las líneas que corresponden a las aristas de todos los objetos del mapa métrico, no sólo esto, también hay que trazar una perpendicular a la línea que une los nodos hacia el punto más cercano a ésta de cada objeto del mapa para evaluar si cabe el robot (con todo y distancia de seguridad). Como es de suponerse lo anterior resulta sumamente costoso en términos computacionales.

Si se tienen *n* nodos libres, al menos es necesario evaluar la conectividad $n(n+1)/2$ veces para tener todas las combinaciones posibles entre nodos. Si para cada par de nodos se debe verificar la intersección con las 4 caras de *m.* objetos en el mapa, hay que multiplicar el factor anterior por 4*m*. Además hay que encontrar el punto más cercano de cada objeto a la trayectoria del robot y medir la distancia perpendicular a dicha trayectoria, sumando 2*m* al valor anterior de 4*m* operaciones, es decir, 6*m* operaciones por cada par de nodos (suponiendo que pudiera hacerse en tiempo constante cada operación). Finalmente se tienen 6*m* operaciones por cada pareja de nodos evaluados. Lo anterior da una complejidad:

$$6m\frac{n(n+1)}{2} = O(mn^2) \tag{6.9}$$

Suponiendo que el número de objetos *m.* se acerque al número *n* de nodos libres de la red, se tendría $O(n^3)$ que resulta bastante lento en términos generales.

6.4.2 Optimización del Algoritmo de Conectividad

En primera instancia se programó el método anterior y se probó con el ambiente de trabajo del robot (malla de 33×40 nodos = 1,320 nodos y 890 nodos libres) como resultado en 35s el algoritmo ha encontrado la matriz de conectividad entre los nodos (en un procesador Power PC a 1 Ghz).

Para hacer más rápido el algoritmo de creación de la red de conectividad entre nodos, implementamos un método de ocultamiento, mediante el cual se toma el primer nodo libre del mapa como origen y el primer objeto en el mapa métrico. A continuación se determinan los ángulos de visión máximo y mínimo (contemplando la distancia de seguridad) de acuerdo a las dimensiones del objeto. Una vez que se han calculado dichas ángulos resulta sencillo tomar como destino uno a uno los nodos restantes y, en tiempo constante, determinar si el ángulo formado por la recta *origen-destino* se encuentra entre los límites previamente calculados. De ser este el caso se determina que no existe conectividad entre el nodo origen y el nodo destino.

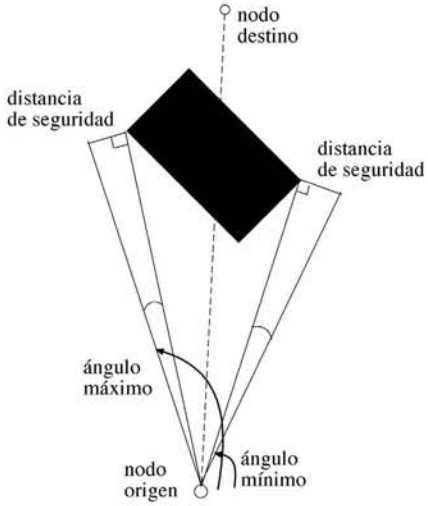

Figura 6.33: Determinación de ángulos de ocultamiento mínimo y máximo.

Si bien esta mejora no disminuye la complejidad de los cálculos, ya que de nueva cuenta para todos los *m.* objetos y al menos $n(n+1)/2$ nodos libres es necesario evaluar si existe ocultamiento, sólo que en esta ocasión resulta mucho más fácil calcular un ángulo y compararlo que hacer intersecciones con las caras del objeto, más trazar perpendiculares a la trayectoria para ver si el robot cabe, con lo que la complejidad disminuye a:

$$m\frac{n(n+1)}{2} = O(mn^2) \tag{6.10}$$

Sin embargo, de nueva cuenta si *m.* se aproxima a *n*, la complejidad es $O(n^3)$. Una ventaja de este método es que no necesariamente es necesario evaluar los *m.* objetos, ya que en cuanto se encuentra el primer objeto que impide la visibilidad hacia el destino, ya no se evalúa el resto de los objetos y se toma el siguiente nodo destino.

Al implementar éste método le llevó 24s encontrar la matriz de conectividad del mapa topológico a la misma computadora.

Por otro lado se observa que como máximo, si el objeto se encuentra muy próximo al nodo, oculta la región entre –90 y 90 grados, por lo tanto es posible descartar de la evaluación, todos los nodos que se encuentren a la izquierda de esta línea u horizonte de visión, es decir, que no se encuentren en la dirección del objeto.

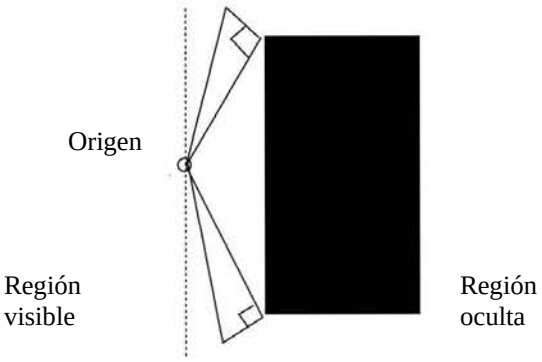

Figura 6.34: Determinación de máxima zona visible y oculta.

Esta adaptación hace que el algoritmo encuentre la conectividad en sólo 8s.

Finalmente una última consideración puede hacerse. Si la distancia del nodo origen al objeto es mayor que la distancia entre los nodos origen-destino, no es necesario hacer ningún cálculo y se descarta el objeto. Con esta última adaptación se redujo finalmente el tiempo de cálculo a 4s, el cual puede considerarse aceptable si se hace fuera de línea.

Como podrá observarse la reducción del tiempo de procesamiento obedeció principalmente a la simplificación de las operaciones y a limitar considerablemente los casos en los que hay que realizar cálculos. Sin embargo, no hay que olvidar que la

complejidad sigue siendo $O(n^3)$ por lo cual los ambientes con mayores dimensiones que las que se han definido para el mapa topológico (33×40 nodos) requerirán de mejores estrategias de optimización que reduzcan la complejidad al menos a $O(n^2 \log n)$ lo cual puede realizarse mediante programación dinámica y/o aprovechando que las pendientes se encuentran de alguna forma ordenadas por la disposición espacial de los nodos en forma de malla.

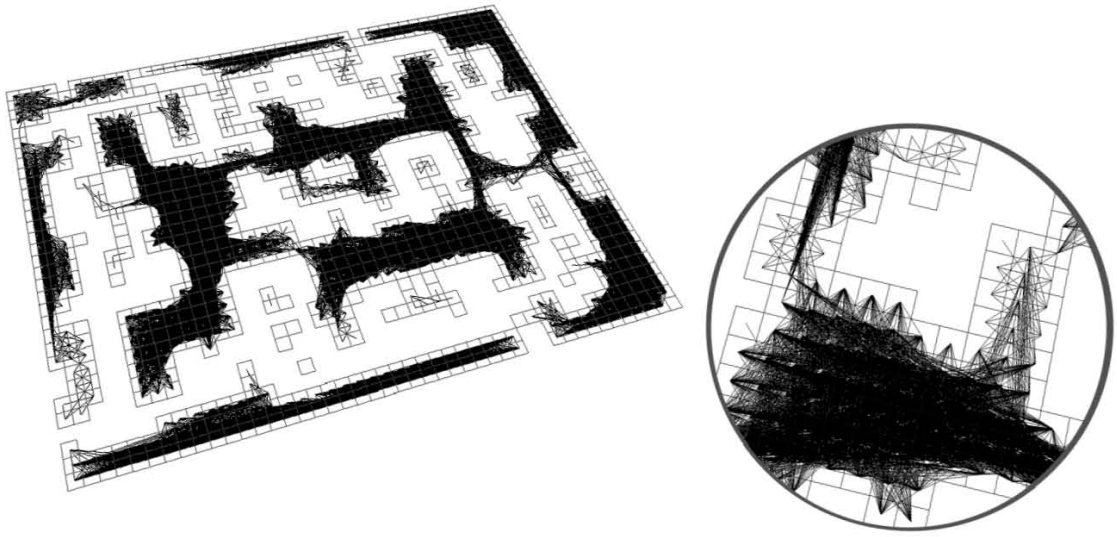

Figura 6.35: Mapa topológico y red de conectividad.

6.5 Determinación de las Zonas (Habitaciones) del Mapa

La noción de "zona" como sinónimo de habitación es uno de los problemas relacionados con la creación de mapas. Sabemos que dicha relación tiene que ver más con el concepto de utilidad o fin que se le da a la misma, que con una relación puramente espacial. Sin embargo, los seres humanos tendemos a agrupar grandes áreas dentro de un mapa que se encuentran delimitadas con barreras tales como paredes o escritorios, de tal suerte que es posible asociar el concepto de zona con un área libre que rebase ciertas dimensiones, mientras que si dicha área es angosta (digamos menor a cuatro veces el ancho del robot) se denominó "pasillo". Pues bien, para que un robot móvil pueda tener una correspondencia entre lo que se ha llamado habitación y su representación espacial deberá ser capaz de encontrar áreas grandes navegables e identificarlas mediante una etiqueta que puede corresponder con el nombre que los seres humanos le asignamos a la habitación. De esta forma se comenzó a crear un mapa conceptual y se estableció una correspondencia entre dicho concepto y una zona (o el centroide de la misma) dentro del mapa métrico.

Desde el punto de vista de cálculo, es posible decir que el proceso de encontrar las zonas del mapa se relaciona directamente con la localización de grandes áreas navegables dentro del mapa métrico o topológico. Se sabe que el mapa topológico utilizado (en forma de malla) indica las posiciones (nodos) libres dentro del espacio de trabajo. Si se considera

el centroide (*x*,*y*) de la celda asociada a cada nodo libre de la red como un punto, el proceso de encontrar las zonas libres del mapa se reduce nuevamente a encontrar los grupo o clusters asociados con las nubes de puntos correspondientes a los nodos libres (navegables) del mapa topológico.

De esta forma puede verse que la red de agrupamiento borrosa anteriormente desarrollada puede ser utilizada de nueva cuenta, ahora en una escala mayor, para encontrar las zonas libres del mapa. Sin embargo, como sabemos dicha red opera asignando un valor borroso a cada punto en función de su confiabilidad (definida como una función de su cercanía con el robot). Sin embargo, en esta ocasión es necesario modificar dicha función de pertenencia ya que, de nueva cuenta, no todos los centroides de las celdas libres serán susceptibles de ser considerados como posibles candidatos a centroides o cluster de la zona. Es necesario eliminar los pasillos estrechos como posibles zonas del mapa. Para lograr lo anterior ahora la función de confiabilidad o pertenencia asociada a cada punto o centroide estará relacionada con el número de celdas libres adyacentes a la misma, es decir, si una celda posee todas sus celdas vecinas navegables, es mucho más susceptible de ser tomada en cuenta (confiable) como una posible zona libre del mapa, que si se tiene una cuyos vecinos inmediatos por ejemplo superior e inferior se encuentran ocupados o no son navegables (lo que significa que se trata de un pasillo angosto).

Se asignó una función de pertenencia normalizada que proporciona un valor de pertenencia igual a 1 para una celda navegable cuyas celdas adyacentes son navegables en su totalidad (ocho en este caso). Por cada celda vecina no-navegable el valor de pertenencia disminuye en 1/3, lo que significa que como máximo las celdas navegables podrán tener dos vecinas ocupadas para poder influir en el agrupamiento.

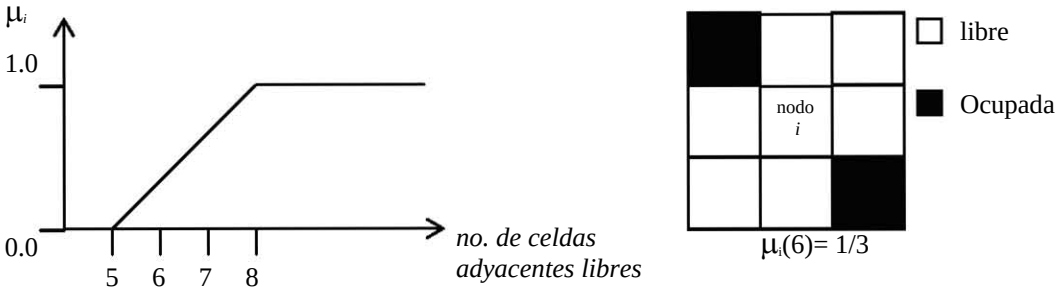

Figura 6.36: Función de pertenencia “confiabilidad” para las zonas libres del mapa y ejemplo.

Finalmente se debe aumentar el radio de influencia de las neuronas utilizado en la red borrosa con agrupamiento bajo demanda, al valor que se desee tomar como radio de las zonas libres (1.4 m). Cabe aclarar que dicho radio de influencia dependerá de lo grande que deseemos que sean dichas zonas libres y de las dimensiones del ambiente real. Por tanto, dicho radio de influencia deberá ser determinado experimentalmente de acuerdo al número aproximado de zonas que se desee obtener.

Siguiendo esta convención se crea una lista de puntos (*x*,*y*) correspondientes a los centroides de las celdas navegables y se asignan valores de pertenencia a cada uno. Cabe señalar que no se toman en cuenta las dos primeras y últimas filas y columnas de nodos

libres ya que en su mayoría son zonas navegables externas al ambiente, generadas debido al desconocimiento de las dimensiones reales del ambiente:

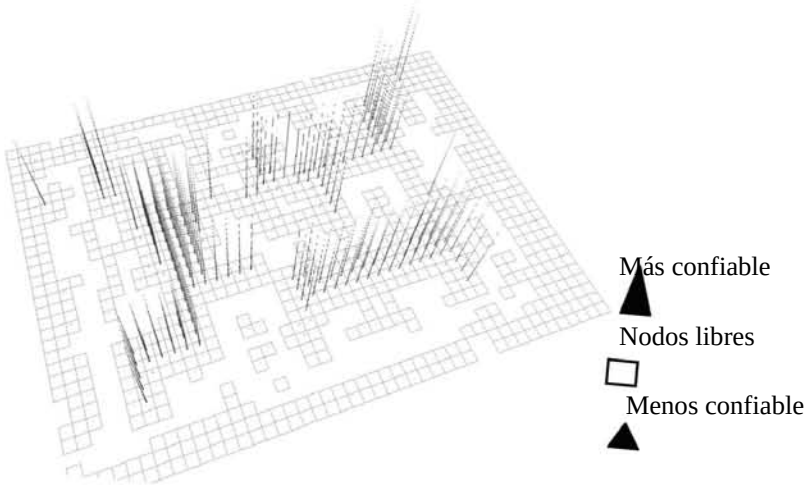

Figura 6.37: Puntos correspondientes a los centroides de los nodos libres del mapa topológico y nivel de confiabilidad de cada uno, utilizados para encontrar las zonas del mapa.

Ahora, de nuevo se realiza el entrenamiento de la red, en esta ocasión la zona de influencia de cada cluster ha sido aumentada a siete veces su tamaño original obteniéndose la ubicación de las zonas encontradas y se utilizó el mismo algoritmo de generación de clusters bajo demanda.

Finalmente es posible eliminar zonas externas al mapa evaluando la conectividad entre el nodo en el que se encuentra el robot (si se encuentra localizado dentro del mapa) y el centroide de cada una de las zonas encontradas. Mediante este método automáticamente se eliminarán todas las zonas exteriores que no pueden ser alcanzadas mediante alguna trayectoria.

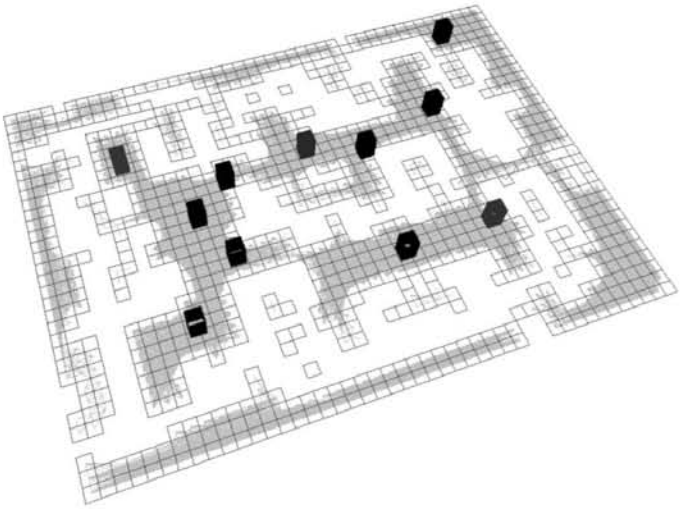

Figura 6.38: Centroides de las zonas encontradas mediante agrupamiento de nodos navegables.

6.6 Navegación en el Ambiente

Una vez que se han generado los mapas métrico y topológico, se ha generado la matriz de conectividad y se han localizado las zonas libres del mapa, es posible hacer que el robot se dirija a una zona (habitación) específica. Para lograr lo anterior el robot deberá ubicar su posición actual dentro del mapa, determinar las coordenadas (*x*,*y*) del centroide (cluster) correspondiente a la zona a la cual se debe dirigir (equivalente al comando "Robot, ve a la sala.", si sabemos qué zona dentro del mapa corresponde al término "sala"), encontrar una trayectoria que lo lleve desde su posición actual hasta la posición destino solicitada y recorrer dicha trayectoria.

Se programó el algoritmo de Dijkstra [Dijkstra 59] para encontrar la ruta más corta entre la posición inicial y final, y que para hacer que el robot recorra la ruta utilizamos el método de campos potenciales (capítulo 2), tomando como destino temporal cada nodo de la trayectoria encontrada. Una vez que un nodo es alcanzado se toma el siguiente nodo dentro de la trayectoria como destino, hasta que se alcanza la posición final. Lo anterior ayuda al robot real a evitar obstáculos inesperados en su recorrido al tomar en cuenta las lecturas de sus sonares en cada paso, hasta llegar a su meta temporal.

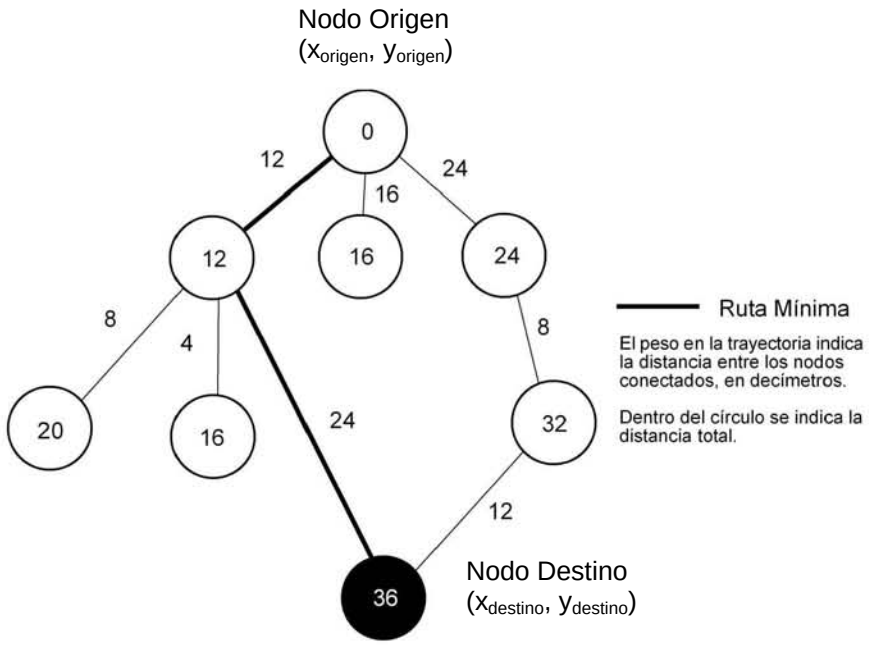

Figura 6.39: Ejemplo de búsqueda con el algoritmo de Dijkstra.

Capítulo 7

Auto Localización Mediante Mapas o Vistas Locales

7.1 Localización con Redes Autoasociativas

Como la red utilizada en la sección 5.8 resultó ineficiente, se decidió ahora probar con una red de Hopfield (o varias redes de Hopfield para ser precisos), tratando de entrenar una red por cada nodo navegable del mapa para que memorizara varias vistas o mapas locales correspondientes a diferentes orientaciones dentro de dicho nodo.

El objetivo era presentar una vista local generada por el robot en una ubicación desconocida a las diferentes redes de Hopfield y determinar qué red responde con el patrón memorizado que mejor se parezca (es decir, cuya función de energía de la red presente el valor mínimo), indicando con ello el nodo y la orientación real del robot y filtrando el ruido provocado por objetos desconocidos.

7.1.1 Creación del Patrón o Vista Local

Se utilizaron los 16 sonares del robot para realizar una lectura de los objetos alrededor del robot y, con base en dichas lecturas y tomando el frente del robot con una orientación supuesta de 90 grados, se determinó dentro de una matriz de 9×9 celdas (centrada en el robot y con cada celda del mismo tamaño que las utilizadas para el mapa topológico) aquella correspondiente a la lectura del sonar (si es que tal medición cae dentro del espacio comprendido por la matriz). Una vez hecho lo anterior fue posible marcar con 0 la celda ocupada por la medición y con 1 las celdas libres (Fig. 7.1 izquierda).

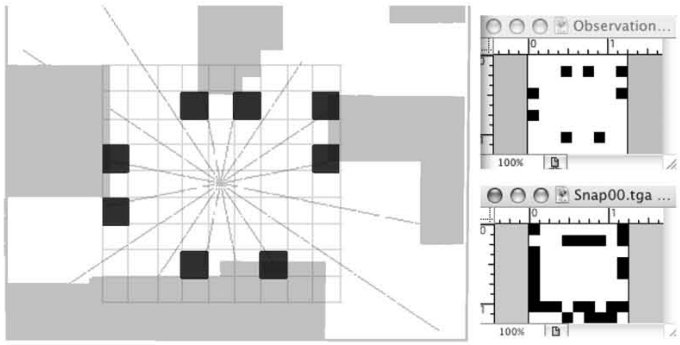

Figura 7.1: Generación del patrón con una vista local del ambiente por barrido.

Si se hace que el robot gire hasta dar una vuelta completa sobre su centro mientras se repite el procedimiento anterior, es posible generar una vista local más completa que sirve como patrón de búsqueda (Fig. 7.1 derecha inferior).

7.1.2 Obtención del Conjunto de Patrones de Entrenamiento para un Nodo

Tómese el caso de un nodo con el que se desea entrenar una red de Hopfield. Una opción para la generación de los patrones de entrenamiento sería llevar al robot al centroide de cada nodo navegable y repetir el proceso anterior de creación de vista local para varias orientaciones iniciales por cada nodo, hasta completar los 360 grados, sin embargo el tiempo total empleado en realizar tal labor resultaría considerablemente elevado. Por otro lado se tiene el mapa topológico que indica cuáles nodos son navegables y su posición dentro del ambiente. Si se toma la información de dicho mapa como base para generar los patrones de entrenamiento, es posible calcular el patrón que el robot debería generar para cada posición y orientación determinadas dentro del ambiente. Lo anterior puede hacerse fácilmente utilizando métodos de rotación de imágenes si se considera a cada patrón como una imagen binaria de 9×9 píxeles, con lo cual es posible generar automáticamente todos los patrones de entrenamiento deseados para cada nodo libre del mapa topológico.

La Figura 7.2 (izquierda) muestra una representación del mapa topológico en forma de imagen binaria, en la parte superior derecha muestra una vista local generada a partir del mapa topológico y debajo de ella la observación real generada por el robot.

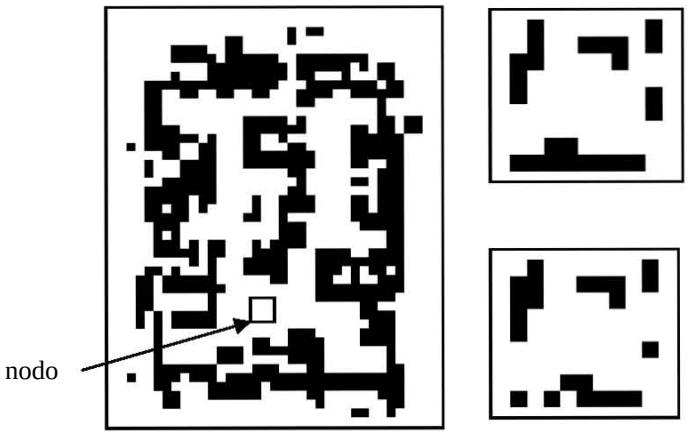

Figura 7.2: Imagen binaria del mapa topológico (izquierda),
observación generada para cierto nodo (arriba der.) y observación real (abajo der.).

7.1.3 Entrenamiento de la Red

Se implementó una red de Hopfield y se generaron con el método anterior 16 patrones de vistas (para el nodo marcado con un recuadro en la Figura 7.2) con diferentes orientaciones girando en sentido antihorario desde 90 grados y hasta cubrir una vuelta completa. A continuación se muestran lo patrones generados:

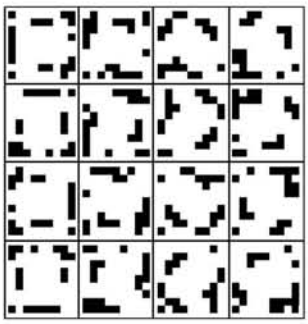

Figura 7.3: Diversos patrones generados para cierto nodo libre del mapa topológico.

La red neuronal implementada recibe como entrada un vector de 81 posiciones (9×9) de tipo binario [1,0], el tipo de función de activación utilizado fue el escalón unitario (*signo de x*). Como método de entrenamiento debimos descartar de inicio el método de Hebb, ya que permite almacenar hasta $0.15n$ patrones (en este caso $0.15\times81 = 12$ patrones) que resulta insuficiente si se desea que la red almacene los 16 patrones generados. En lugar del método de Hebb se utilizó el método de la matriz pseudoinversa utilizando el método de Greville [Kohonen 89] que establece como regla de aprendizaje la siguiente:

$$\overline{W}_{(i+1)} = \overline{W}_{(i)} + \eta\,(d - (\overline{W}_{(i)} \bullet \overline{X}_{(i)})) \qquad (7.1)$$

donde η es el ritmo de aprendizaje, d es la salida deseada y $\overline{X}$ es el vector de entradas.

Al probar el método con un conjunto de patrones (números del 0 al 9) los resultados no fueron totalmente satisfactorios, ya que la red sólo pudo memorizar correctamente unos cuantos números. Por otro lado, la ecuación 5.6 es sumamente parecida a la regla delta para las neuronas de salida de una red *feed-forward*. Como en este caso (red de Hopfield) sólo existe una capa de neuronas (cada neurona es tanto de entrada como salida) es posible aplicar el método de retropropagación para entrenar la red (en realidad sólo se aplica para la última capa), sin embargo para poder utilizarlo es necesario sustituir durante el entrenamiento la función escalón por una función sigmoide (se utilizó la ecuación 3.7). Una vez hecho lo anterior es posible aplicar la regla delta para la función sigmoide (3.7):

$$\overline{W}_{(i+1)} = \overline{W}_{(i)} + \eta\,(d - f)(1 - f)\,\overline{X} \qquad (7.2)$$

donde η es el ritmo de aprendizaje, d es la salida deseada, f es la salida de la neurona y $\overline{X}$ es el vector de entradas.

Como conjunto de entrenamiento se colocaron los patrones a memorizar como vectores tanto de entrada como de salida de la red (ya que para cada patrón a almacenar se pretende que la red nos proporcione ese mismo patrón a la salida). Una vez concluido el entrenamiento restituimos la función escalón.

7.1.4 Resultados del Entrenamiento

La Figura 7.4 muestra los resultados del entrenamiento de la red con patrones binarios (números del 0 al 9) para probar su correcto funcionamiento:

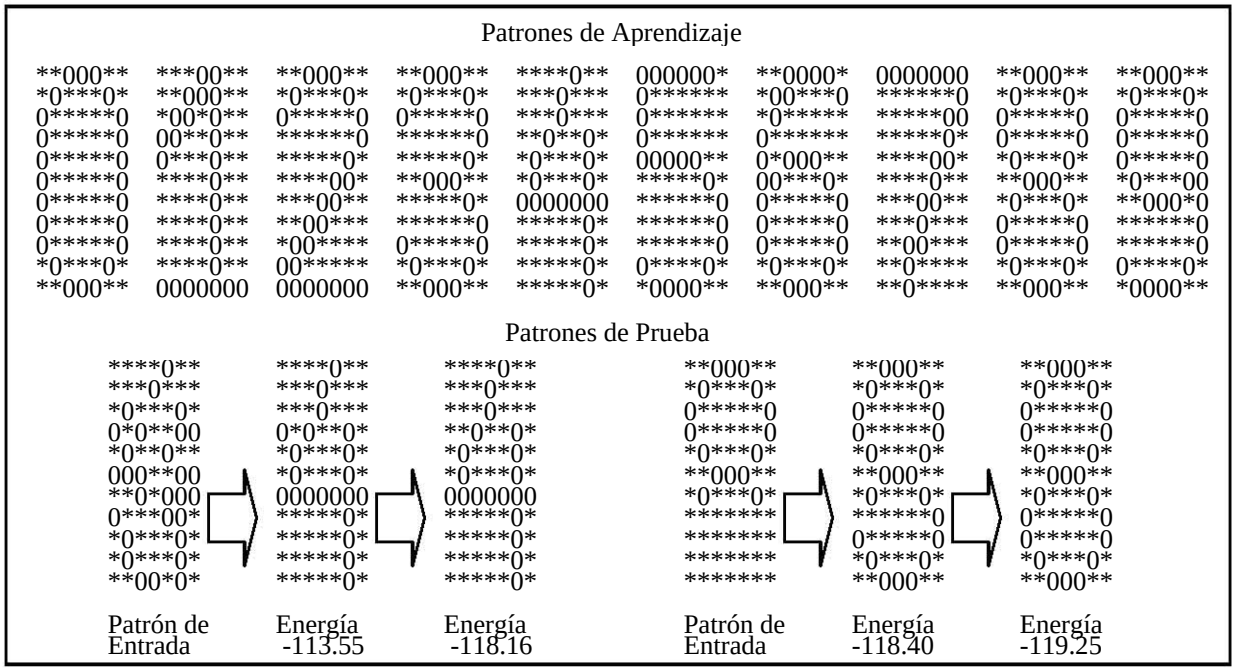

Figura 7.4: Prueba de la red de Hopfield con patrones numéricos.

La red parece comportarse perfectamente, sin embargo al tratar de entrenarla con los 16 patrones generados para las vistas se observó que la red no pudo entrenar correctamente y presentó un error de entrenamiento cercano al 60% (Fig. 7.5). Este resultado, aunque desalentador, era de esperarse, ya que para el caso de los patrones generados para distintas orientaciones de un nodo se observa que existe mucha similitud entre ellos a pesar de que estamos variando la orientación entre cada patrón. La diferencia entre ellos no es suficiente para permitir que la red aprenda a diferenciarlos, por lo cual siempre presenta la misma salida errónea.

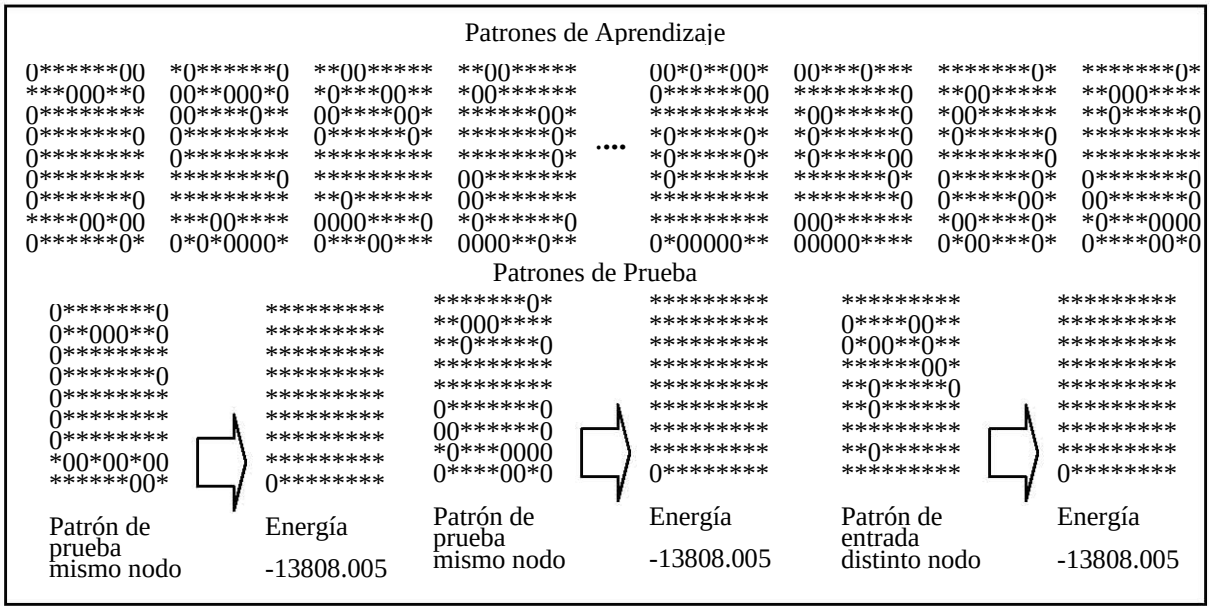

Figura 7.5: Prueba de aprendizaje de la red de Hopfield con patrones generados para un nodo.

Dado el problema anterior, es posible concluir que el uso de este tipo de red para predecir la orientación resultó insatisfactorio debido a la gran similitud entre los patrones que se desea que la red aprenda, por consiguiente se tendrá que evaluar algún otro método que permita buscar el patrón de la vista local generada por el robot, dentro de la representación gráfica binaria del mapa topológico (Fig. 7.2).

7.2 Localización Mediante Correlación de Imágenes

Es posible hacer uso de la representación gráfica binaria del mapa topológico y la vista local generada por el robot en la posición desconocida, para tratar de encontrar la zona de mayor correspondencia entre ambas, es decir, es posible tratar de ubicar como imagen la observación local del robot dentro de la imagen del mapa topológico (ya que ambas poseen la misma escala), para lograr lo anterior lógicamente no bastará con buscar la vista local con una sola orientación, sino que será necesario generar a partir de esta diversas vistas locales con diferentes orientaciones y buscar la máxima coincidencia.

El procedimiento anterior se asemeja al armado de un rompecabezas en donde la vista local representa la pieza que queremos colocar. Para encontrar la posición correcta se debe girar varias veces la pieza hasta encontrar el lugar donde mejor se acople. Una vez hecho esto podremos dar un estimado de la posición y orientación original desde donde se generó la vista local, que corresponde con la posición y orientación del robot real.

Para buscar la zona de mayor coincidencia de la vista local generada por el robot y la imagen del mapa topológico es posible utilizar el método conocido como *correlación de imágenes*, ésta es utilizada como medida de similitud en el contexto de *coincidencia* o *correspondencia de patrones* o *planillas* [González 02], y se define como sigue:

$$(h \oplus I)(x,y) = \int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} h(u,v)I(x+u,y+v)\partial u \partial v \tag{7.3}$$

En el procesamiento de imágenes, la versión discreta y bidimensional de la correlación es:

$$(h \oplus I)(x,y) = \sum_i \sum_j h(i,j)I(x+i,y+j) \tag{7.4}$$

donde I es la matriz imagen original y h es la función de ponderación de la correlación también llamada filtro o *kernel*. Dependiendo de las características del filtro o kernel y del tamaño del mismo será la imagen resultante.

Para encontrar la posición del robot se debe desplazar el patrón o kernel sobre la imagen y se busca la ubicación con máxima coincidencia utilizando la correlación.

Los valores altos de correlación indican una buena correspondencia entre la imagen I y el patrón h.

De acuerdo con lo anterior sería suficiente con tomar la imagen *I* del mapa topológico previamente generada y aplicar la correlación con el patrón *h* buscado (vista local generada por el robot) para encontrar los puntos de la imagen resultante con la mayor coincidencia. Sin embargo, al hacer esto se observa un problema serio: si se toma directamente el patrón a buscar como kernel y se aplica la correlación, aquellas zonas de la imagen con mayor correspondencia aparecerán como puntos con mayor intensidad en la imagen resultante sin importar que no exista coincidencia alguna en el resto de los píxeles, ya que el resultado de la correlación para cada píxel será la suma de la multiplicación de los valores binarios del kernel (1 o 0) por los valores binarios (1 o 0) del mapa cuando el kernel se localiza en dicho píxel. Dicho de otra forma, el procedimiento anterior puede indicar que el robot se encuentra justo en la posición ocupada por algún objeto, obviamente no navegable, si en su mayoría el patrón buscado coincide con el mapa en dicha posición.

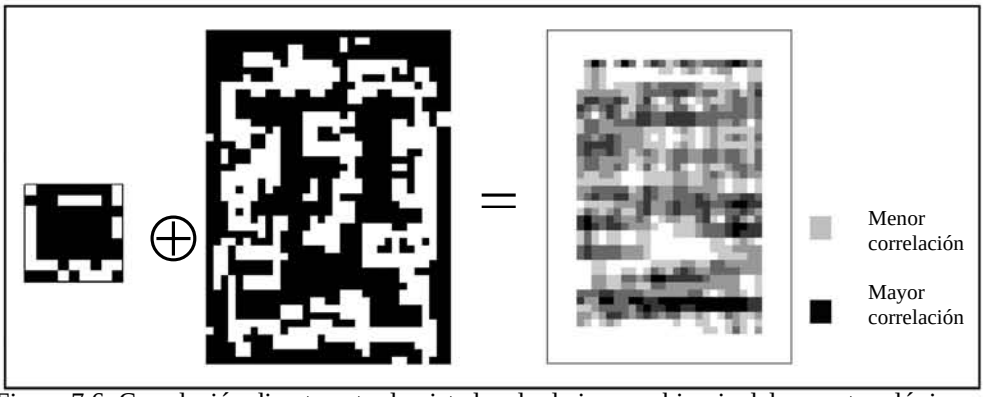

Figura 7.6: Correlación directa entre la vista local y la imagen binaria del mapa topológico.

Como puede apreciarse en la figura anterior el método entrega posiciones que en la práctica es imposible que el robot pueda ocupar ya que, si bien en gran medida el patrón buscado coincide con el mapa para las posiciones encontradas, no toma en cuenta que en la parte central de la posición encontrada se encuentren objetos o no exista coincidencia alguna con el patrón buscado.

7.2.1 Determinación del Filtro Óptimo

Un aspecto importante en la localización es que las ubicaciones encontradas deben estar libres de objetos. Por lo tanto, determinamos modificar el kernel para penalizar, las ubicaciones donde el patrón coincide con el mapa en una ubicación errónea. Se probó con varios kernels y finalmente se utilizó un kernel gaussiano que penaliza negativamente el resultado de la correlación para ubicaciones donde existan objetos en la parte central. Para las celdas coincidentes se le asignó un valor de 2.0 al kernel. El resultado de la correlación se divide entre dos veces el número de celdas ocupadas del kernel para Conforme a lo anterior, a pesar de que exista una coincidencia del 100% entre el patrón y la región del mapa encontrada bastará que exista un objeto en el mapa en la posición central del kernel para que la correlación proporcione un valor negativo (los valores negativos se hacen iguales a cero). Por ejemplo, si se tiene un patrón con 24 celdas ocupadas (Fig. 7.7) y coincide al 100% con una región del mapa pero se tiene un objeto al centro de la región

encontrada, se tendría un valor de correlación igual a $(24\times255\times2 - 255\times27)/48 = 111$ en lugar del valor 255 que sería el valor máximo de coincidencia para ese patrón.

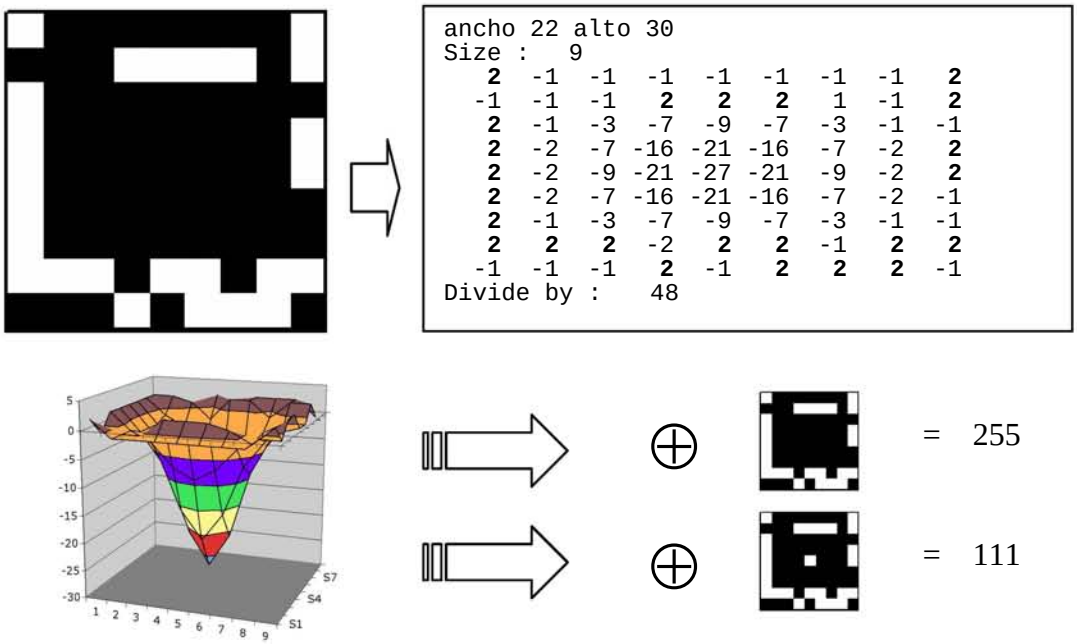

Figura 7.7: Estructura del kernel modificado y ejemplos de correlación en 2 posiciones.

Como resultado de la modificación anterior se observa que una gran cantidad de posiciones erróneas disminuyen su valor (Fig. 7.8) y sólo un reducido número de posiciones poseen altos índices de correspondencia.

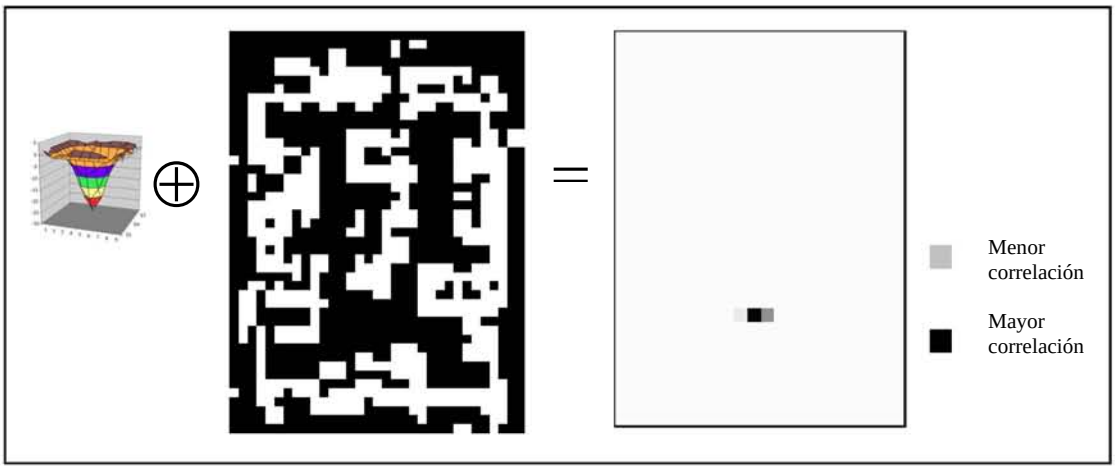

Figura 7.8: Correlación entre kernel modificado e imagen binaria del mapa topológico.

Una vez hecho lo anterior, se obtiene una imagen cuyos valores (entre 0 y 255) representan una hipótesis de localización del robot para cada posición o nodo (píxel de la imagen resultante). Sin embargo, dado que se desconoce la orientación original del robot al momento de generar la vista local utilizada en la correlación, no es posible garantizar que la zona de máxima coincidencia se dé en la ubicación correcta para la vista local, ya que para que esto suceda tanto el mapa como el patrón deben poseer la misma orientación. Lo

anterior constituye una ventaja si se considera que es posible generar a partir de la vista local diferentes patrones con distintas orientaciones. Si se efectúa la correlación con cada uno de los patrones generados, dado que se conoce el grado de rotación de cada uno de ellos con respecto al original, al momento de encontrar la máxima zona de coincidencia entre todas las imágenes generadas, se podrá saber cuánto se ha girado la vista local original para encontrar la máxima coincidencia, es decir, la orientación del robot con respecto de la orientación del mapa y, por consiguiente, la orientación real del robot dentro del ambiente real de trabajo. Sin embargo, en este punto es necesario definir una estrategia para seleccionar la posición final que se tomará como válida a partir de un conjunto de imágenes, cada una de ellas con las hipótesis de localización y orientación del robot.

7.2.2 Combinación Borrosa de las Hipótesis de Localización

El primer criterio utilizado para seleccionar la posición y orientación del robot fue encontrar el máximo valor, entre la correlación de los patrones generados rotando la vista local y la imagen del mapa topológico. Al encontrar el píxel con el máximo valor de intensidad de entre todas las imágenes automáticamente se conoce la posición de la mejor estimación y si se considera que, cuando el patrón coincide con una zona de la imagen binaria del mapa ambos están orientados hacia 90 grados, como se sabe cuánto se hizo girar al patrón original para alcanzar la máxima coincidencia, dependiendo de la dirección del giro se suma o resta ese valor a 90 grados, determinando así la orientación del robot real.

Al utilizar este criterio se encontró el problema de que en ocasiones se presenta una alta coincidencia del patrón en otras posiciones y orientaciones, inclusive mayor que aquella obtenida en su posición real, lo anterior debido a que por ligeras variaciones o inclusive errores en las lecturas de los sonares la vista local no corresponde al 100% con lo referido en la imagen del mapa (recuérdese que para obtenerlo se utilizó una red de agrupamiento borrosa y no sólo las lecturas en crudo de los sonares).

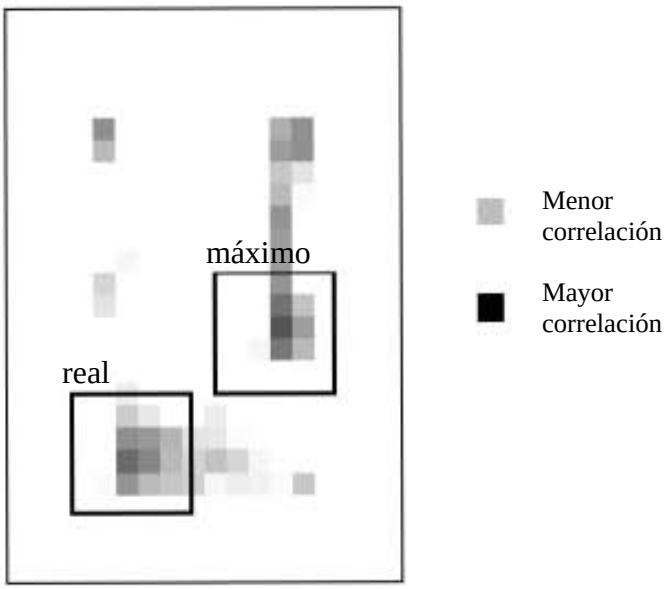

Figura 7.9: Ubicación errónea encontrada por el criterio del máximo.

Al revisar detenidamente los resultados obtenidos se concluye que no basta con buscar el máximo valor de coincidencia, ya que por los errores mismos en las mediciones y la discretización al momento de generar la imagen del mapa topológico y la vista local, se pueden dar coincidencias aleatorias mayores en otras regiones de la imagen. Sin embargo, se observó que, para la ubicación real, no sólo se presenta una alta coincidencia en la orientación exacta, sino en la imagen inmediata anterior y en la inmediata posterior a la orientación real, lo cual nunca sucedió para las localizaciones erróneas. Haciendo uso de esta premisa se decidió utilizar algún método que tomase en cuenta o que mezclara todas las imágenes obtenidas, es decir, todas las hipótesis de localización.

Figura 7.10: Imágenes correspondientes a hipótesis de orientación de 90^o (izquierda) y 112.5^o (derecha).

Si se considera que cada imagen indica con un valor de 0 a 255 para cada píxel la correlación entre el mapa y el patrón girado de la vista local, al normalizar estos valores entre 0 y 1, pueden ser vistos como el valor de la función de pertenencia para cada punto y orientación de una variable borrosa que es posible llamar *ubicación*, de tal suerte que se definir

$$\mu_{ubicación}(x,y,\phi)=\frac{I_{\phi}^{*}(x,y)}{255} \tag{7.5}$$

donde I_{ϕ}^{*} es la imagen resultado de la correlación del mapa I con el kernel girado h que implica una orientación real ϕ.

De esta forma es posible realizar una operación de unión o intersección borrosa píxel por píxel tomando en cuenta todas las imágenes resultado. Dependiendo de la operación elegida se puede hacer que el valor de la función de pertenencia para la ubicación correcta sea mayor que para el resto, si se logra que al combinar las imágenes sucesivas cercanas a la orientación real del robot se intensifique su valor. Por el contrario para las ubicaciones erróneas no bastará con que sólo una de ellas posea un alto valor.

Se generaron 16 imágenes producto de la correlación entre la imagen del mapa y el kernel girado 22.5 grados cada vez. Para combinar las imágenes se utilizó la operación borrosa conocida como suma de Hammacher (ver cap. 4, tabla 4.1) ya que ésta permite aumentar el valor de la función de pertenencia resultante por encima del máximo entre los dos valores operados, lo que asemeja un efecto de amplificación. Cuando alguno de dichos valores es cero, la función simplemente devuelve el valor máximo entre ellos.

	0.00	0.10	0.20	0.30	0.40	0.50	0.60	0.70	0.80	0.90	1.00
0.00	0.00	0.10	0.20	0.30	0.40	0.50	0.60	0.70	0.80	0.90	1.00
0.10	0.10	0.18	0.27	0.35	0.44	0.53	0.62	0.71	0.80	0.90	1.00
0.20	0.20	0.27	0.33	0.40	0.48	0.56	0.64	0.72	0.81	0.90	1.00
0.30	0.30	0.35	0.40	0.46	0.52	0.59	0.66	0.73	0.82	0.90	1.00
0.40	0.40	0.44	0.48	0.52	0.57	0.63	0.68	0.75	0.82	0.91	1.00
0.50	0.50	0.53	0.56	0.59	0.63	0.67	0.71	0.77	0.83	0.91	1.00
0.60	0.60	0.62	0.64	0.66	0.68	0.71	0.75	0.79	0.85	0.91	1.00
0.70	0.70	0.71	0.72	0.73	0.75	0.77	0.79	0.82	0.86	0.92	1.00
0.80	0.80	0.80	0.81	0.82	0.82	0.83	0.85	0.86	0.89	0.93	1.00
0.90	0.90	0.90	0.90	0.90	0.91	0.91	0.91	0.92	0.93	0.95	1.00
1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	DIV#0

Tabla 7.1: Comportamiento de la suma de Hammacher.

Al operar las imágenes, al llegar a las que se encuentran cercanas a la orientación real, dado que las tres posen un alto valor de correlación en la posición real, la imagen resultante de operar todas las imágenes presentará un valor de pertenencia mayor en la ubicación real que en las posiciones de las ubicaciones erróneas.

Finalmente para elegir la posición real del robot se busca el máximo valor en la imagen combinada con la operación borrosa, mientras que una vez conocida la posición, se podrá encontrar la orientación buscando el valor máximo entre todas las imágenes exclusivamente para la posición (x,y) del píxel determinado como posición del robot.

Figura 7.11 Unión borrosa de las imágenes de localización utilizando suma de Hammacher (izquierda) y posición del robot encontrada (derecha), con la observación local superpuesta (gris).

Capítulo 8

Resultados

8.1 Introducción

Esta sección muestra los resultados de aplicar los métodos descritos en el presente trabajo en dos categorías[1]:

1) Creación del mapa y localización
2) Navegación autónoma

Dentro del primer rubro se muestran las capacidades del robot para crear una representación de su medio ambiente en función de las observaciones. Lo anterior es realizado en dos modalidades:

- Modo guiado
- Modo autónomo

En el primer caso el objetivo es poder evaluar la exactitud del mapa generado utilizando la ayuda de un operador para guiar al robot en su recorrido, mientras que en segundo el robot de forma autónoma debe hacer el trabajo por sí solo.

Como medida de comparación se presentan los resultados obtenidos con redes neuronales artificiales y localización por correlación de imágenes, contra métodos que no hacen uso de estas técnicas.

8.2 El Ambiente de Trabajo

Se acondicionó una sala de un laboratorio de computación como escenario para las pruebas. Para evitar el problema de que el robot se introdujera debajo de las mesas de trabajo éstas se forraron con cartoncillo para que el sonar las detectase, sin embargo se dejaron pequeños huecos (separación de 30 cm.) entre algunas láminas de cartoncillo para observar qué tanto engañaban al robot. Las dimensiones aproximadas del área de trabajo fueron 10 × 8 m y las mesas se acomodaron para dejar una zona navegable con forma de "M." (Fig. 8.1).

[1] Resultados obtenidos durante el *Primer Concurso Mexicano de Robótica* en la categoría de *Robots de Servicio*.

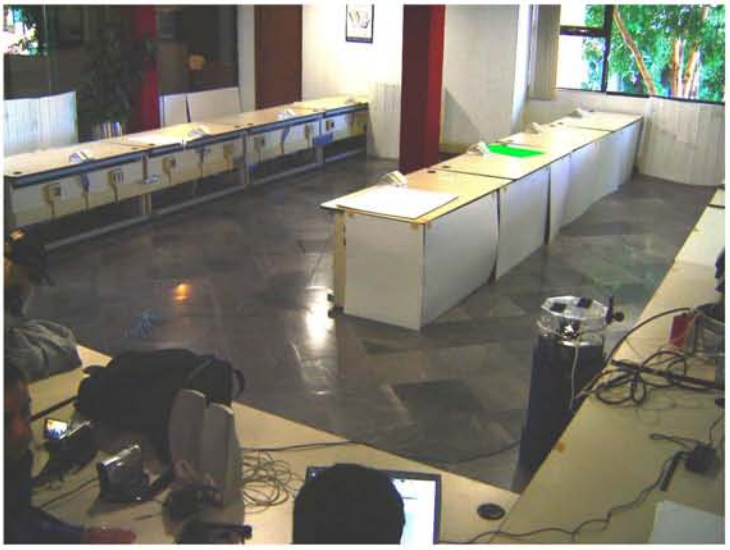

Figura 8.1: Ambiente de trabajo para la pruebas.

Cabe señalar que se dejó sin recubrir una sección de 2 × 2 m del cristal que separa la sala de computación de un pasillo lateral, para observar si afectaba las mediciones del robot.

8.3 Adaptaciones a los Modelos

Haciendo un análisis a los sonares del T×8 (ver cap. 5) fue posible saber que no existiría problema con las superficies de cartón, ya que eran suficientemente gruesas como para permitir la reflexión de la señal de sonar y que, si la señal incidía de forma perpendicular a la superficie de las láminas de cartoncillo, no habría ningún problema en registrar mediciones que indicasen la presencia de las mesas.

La única adaptación que fue necesario hacer en las pruebas preliminares fue aumentar el radio de influencia de la red de agrupamiento borrosa, utilizada para encontrar las zonas o habitaciones del mapa, de 1.4 m a 2.2 m ya que, dadas las dimensiones del área de trabajo, que resultaron un poco mayores a las del Laboratorio de Biorrobótica, el método encontraba aproximadamente 38 zonas. De igual forma se tuvo que aumentar el área utilizada como vista local para la localización por correlación de imágenes, de 1.4 × 1.4 m a 2.2 × 2.2 m ya que, con las primeras dimensiones cuando el robot se encontraba al centro de los pasillos, como la separación entre las hileras de mesas era de 2 m aproximadamente, la vista local generada contenía exclusivamente celdas libres y el robot no se localizaba correctamente.

8.4 Resultados

8.4.1 Modo Guiado

En primera instancia se elaboró el mapa con el método guiado, haciendo avanzar al robot 40 cm. entre cada toma de lecturas sucesivas. El robot se controló de forma remota utilizando una conexión inalámbrica TCP/IP entre la computadora localizada sobre el robot y una computadora de escritorio. Desde dicha computadora de escritorio era posible enviar comandos de movimiento al robot. La Figura 8.2 muestra el mapa métrico elaborado, la red de conectividad y las 14 zonas encontradas (se muestra el centroide da cada zona encontrada con un cubo de distinto color). Las zonas claras representan los objetos en el ambiente de trabajo, como mesas, sillas, etc.

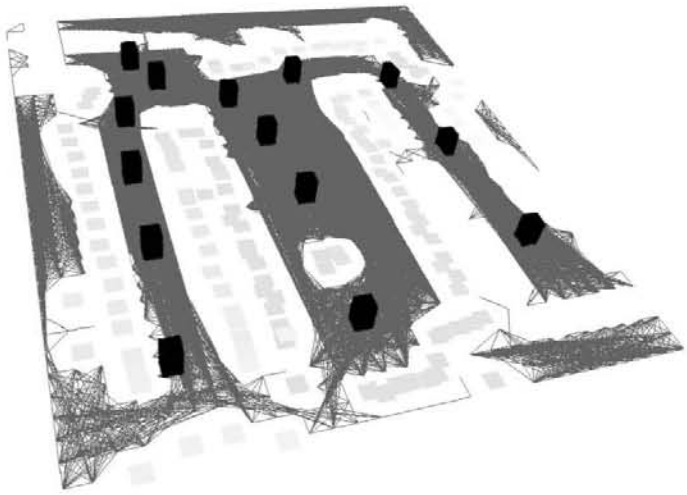

Figura 8.2: Mapa métrico (guiado), red de conectividad y zonas encontradas (14).

Como puede apreciarse en la figura anterior el método encontró una representación bastante acertada del entorno partiendo de la esquina inferior izquierda del mapa, el único problema fue que en algunas zonas faltaron lecturas para evitar que se encontrasen conexiones a través de los objetos y que la superficie navegable resultase enmarcada por una superficie cerrada. Lo anterior se puede superar llevando al robot más cerca de las mesas con un avance menor, digamos de 20 cm. entre cada paso.

8.4.2 Modo Autónomo

A continuación se procedió a utilizar la red de navegación autónoma del robot y generar el mapa métrico, encontrar la red de conectividad y las zonas del mapa. Se partió de la misma posición que para el caso guiado y se dejó que el robot se desplazara libremente por el espacio de trabajo. Se fijó una distancia de 20 cm. entre cada paso del robot para tratar de mejorar el primer mapa elaborado.

En la Figura 8.3 se muestra el recorrido efectuado por el robot, en el que se puede apreciar que hay una zona al fondo del laboratorio donde no ingresa el robot en su recorrido, ya que la red neuronal fue entrenada para evitar zonas angostas (por debajo de 4 veces el diámetro del robot) y por ello la evita.

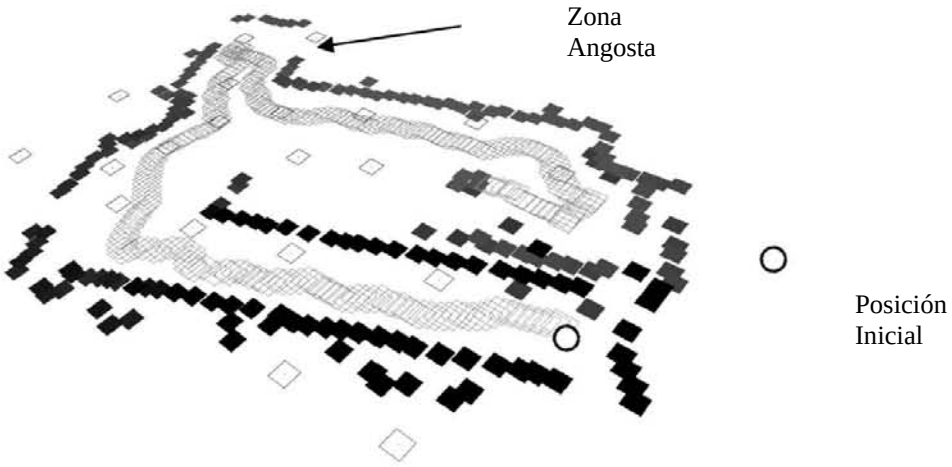

Figura 8.3: Ruta seguida con movimiento autónomo y mapa métrico elaborado.

En la misma figura, al final de la trayectoria del robot, se aprecia que el mismo colisiona contra una columna revestida con hule espuma. En una primera instancia pudiera pensarse que el robot no había detectado la columna, sin embargo al observar los grupos o clusters hallados por el mismo se observa que sí existen mediciones que indiquen su presencia, sin embargo, dado que se trata de un material que absorbe la señal del sonar y que nunca se entrenó la red con ese tipo de material, la red envía directamente al robot hacia el obstáculo.

En la Figura 8.4 el robot partió con una orientación diferente que en el caso guiado donde se alineó con la dirección norte del mapa. Se puede ver que como consecuencia de lo anterior la red de conectividad encuentra nodos libres al exterior de la zona navegable del mapa, sin embargo, si todos los clusters encontrados se encuentran suficientemente cerca los unos de los otros para evitar que el robot pueda pasar entre ellos no existirá una conectividad entre la zona navegable externa y la interna del mapa métrico. En este caso se pueden apreciar algunas aberturas que conectan dichas zonas lo que constituye un problema que puede resolverse disminuyendo el avance o paso del robot a un valor menor para hacer que las lecturas se encuentren mas próximas entre sí.

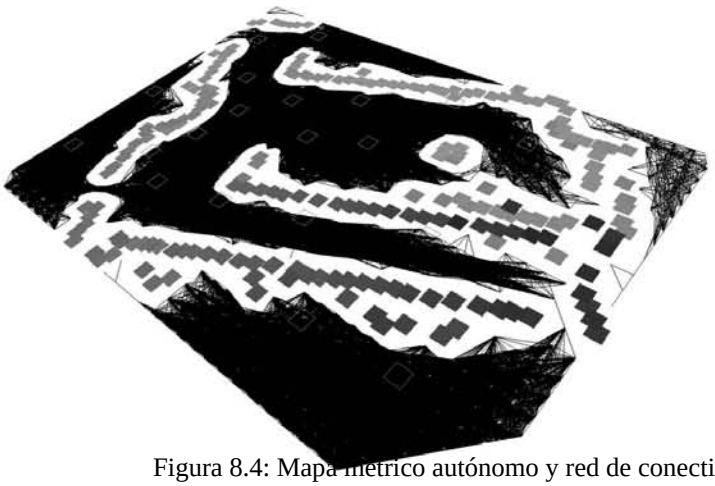

Figura 8.4: Mapa métrico autónomo y red de conectividad.

8.4.3 Auto Localización

Se colocó al robot al centro de la habitación de frente hacia una hilera de mesas y se ejecutó la rutina de auto localización por correlación de imágenes (sección 7.2). Se muestra el resultado en la Figura 8.5. El robot encontró correctamente su orientación, sin embargo se localizó en un nodo a la derecha de su ubicación real (respecto a la orientación mostrada en la Figura 8.5). Si bien el error de localización fue de sólo 30 cm. (lo que se puede considerar bueno), al momento de hacer que rodeara un grupo de mesas, la trayectoria encontrada lo llevó directamente a impactarse contra la esquina de una de ellas, a pesar de utilizar campos potenciales para tratar de evadirla durante varios intentos. Por tal motivo el método empleado para la localización debe ser modificado, permitiendo actualizar o refinar su estimación mientras avanza por el ambiente de trabajo.

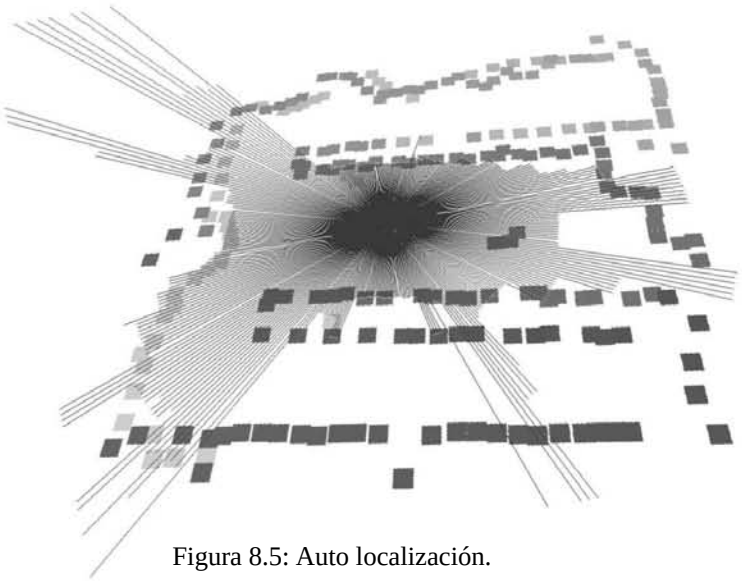

Figura 8.5: Auto localización.

8.4.4 Navegación

Para probar la eficiencia de la navegación utilizada por el T×8 se realizó una prueba con dos robots, el T×8 y otro robot denominado R2 cuyos métodos de creación del mapa, localización y navegación no incorporasen redes neuronales artificiales[2] (Fig. 8.6). Lo anterior con la finalidad de probar la capacidad de ambos para evadir obstáculos inesperados. Para ello se colocó a cada robot a un extremo de la sala y se hizo que ambos se dirigiesen al extremo opuesto, para determinar cuál de los dos podía realizar correctamente la tarea encomendada en el menor tiempo posible. Obviamente existiría un punto en el cual los dos robots se encontrarían frente a frente y se debían evadir mutuamente.

Figura 8.6: Robot R2 y robot T×8.

Una vez creado los mapas métrico y topológico, se asignó una etiqueta a cada zona, por ejemplo, la zona no. 1 es la *cocina*, la 2 es la *sala*, etc. y cada zona corresponde al centroide de uno de los clusters de los nodos libres que la red borrosa encuentra.

El comando dado al robot de forma escrita, a través de la una interfaz de usuario elaborada para facilitar las tareas de control fue: "Robot ve a la cocina" ("*robot, go to the kitchen"* en idioma inglés). Como resultado de la orden anterior el T×8 ejecutó el algoritmo de Dijkstra y encontró la ruta mas corta entre su posición actual y el centroide de la zona destino (Fig. 8.7).

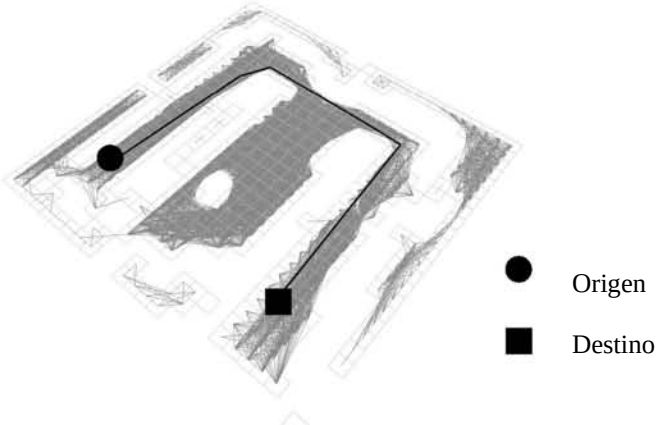

Figura 8.7: Ruta encontrada para llegar al extremo opuesto del ambiente de trabajo.

[2] Robot propiedad del ITESM campus cuernavaca.

Una vez encontrada la ruta óptima, el T×8 trató de seguir dicha trayectoria utilizando campos potenciales (ver sección 2.4.3). En cada paso se fija como destino temporal un punto dentro de la trayectoria a cierta distancia del robot (30 cm.) y se calcula una fuerza de atracción para dirigir al robot a dicha posición y una fuerza de repulsión por cada objeto detectado con los sonares. Una vez que el robot ha avanzado cierta distancia (15 cm.) se fija un nuevo punto destino sobre la ruta a la misma distancia del robot (30 cm.), se repiten los cálculos y se determinan los movimientos sucesivos hasta que el robot alcanza la posición final. El método anterior obliga al robot a seguir la trayectoria encontrada pero lo hace susceptible a las mediciones de sus sonares y si se presenta un objeto extraño dentro de la trayectoria, lo tratará evitar.

Como el algoritmo de movimiento con campos potenciales opera rápidamente y ya se ha calculado previamente en un lapso realmente corto (unos pocos milisegundos) la ruta a seguir por el robot, en poco tiempo el T×8 alcanzó el extremo opuesto y gracias a los campos potenciales logró evadir al R2 (mientras éste trataba de encontrar su posición muy cerca de su punto de partida, ya que un movimiento erróneo lo llevó a colisionar contra una mesa). La tarea anterior fue completada con éxito haciendo muy pocos ajustes a los parámetros utilizados por los modelos descritos en el presente trabajo.

La Figura 8.8 muestra la interfaz de usuario elaborada para enviar comandos al robot durante la prueba.

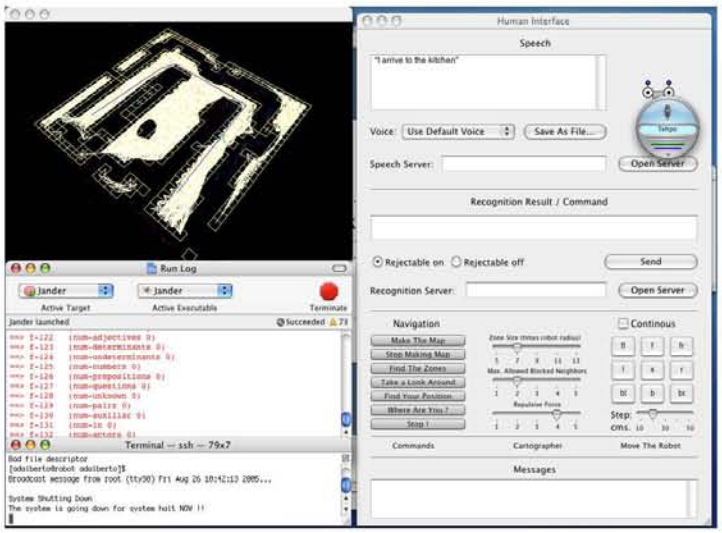

Figura 8.8: Interfaz de usuario del programa de navegación.

8.5 Comparación de Resultados

Para crear el mapa métrico el R2 utilizó un barrido simple con sonares y láser. En cada punto de parada del robot, éste realiza una lectura con sus sensores, gira muy levemente entre cada toma y determina la dirección de avance buscando la distancia más lejana. Utilizó una malla con celdas de 5×5 cm. por lo que el mapa generado (fuera de línea) tuvo una buena resolución, sin embargo el tiempo que hubo que esperar fue considerable (10 min. aproximadamente).

Como resultado el R2 generó dos mapas, uno con las lecturas de los sonares y otro con las lecturas láser. Para unir ambos mapas sólo se sobreimpusieron las lecturas. En cuanto al movimiento, el R2 se desplaza por el ambiente mediante pasos cortos (5 cm.) por lo que tarda en recorrer el espacio de trabajo alrededor de 40 minutos. En cuanto a la localización, el R2 realiza una búsqueda de discontinuidades en las superficies de los objetos, las cuales compara contra su mapa métrico para encontrar su posición y orientación en función de la máxima coincidencia entre discontinuidades. Esta localización es bastante precisa (5 cm. de error), pero sumamente lenta y depende de que existan dichas discontinuidades.

La Tabla 8.1 hace una comparación entre los métodos evaluados:

Método Propuesto	Método no Neuronal	Resultado
a) Creación del Mapa Métrico Creación del mapa utilizando sonares y Red Borrosa de Agrupamiento	Creación del mapa utilizando barrido con sonares y láser, uniendo ambos mapas con un OR simple.	La red borrosa permite generar el mapa en tiempo real, mientras que el algoritmo de barrido lo hace fuera de línea (aproximadamente en 10 minutos).
b) Movimiento autónomo del Robot Red neuronal entrenada para seguir paredes	El robot encuentra la mayor distancia durante el barrido y sigue esa distancia, siempre y cuando no regrese en la misma dirección (algoritmo determinístico).	La red neuronal llevó al $T\times 8$ a colisionar con un objeto de un material absorbente a la señal del sonar e hizo que no se introdujese por un pasillo relativamente estrecho. El R2 al seguir la mayor distancia, chocó contra una mesa al existir una separación entre las placas de cartón.
c) Mapa Topológico Utilizó una malla con celdas de 30×30 cm. donde cada celda representa un nodo	Utiliza también una malla, pero cada celda es de 5×5 cm.	El mapa métrico y topológico es más fino cuando se utilizan celdas pequeñas pero el procesamiento tarda mucho más tiempo y debe hacerse fuera de línea.
d) Red de Conectividad Se calcula una vez elaborado el mapa métrico, aproximadamente en 30 s.	No se elabora la red de conectividad	Resulta mucho más práctico calcular fuera de línea la conectividad entre nodos, fundamental para el cálculo rápido de trayectorias.
e) Auto localización Por correlación de imágenes entre el mapa del medio y la observación local y combinación borrosa de las localizaciones.	Búsqueda de discontinuidades. Se efectúa un barrido en la posición actual y se buscan las discontinuidades en las superficies. Se comparan contra el mapa global y se encuentra la posición y orientación.	Ambos métodos trabajan bien, sin embargo la localización por búsqueda de discontinuidades presentó menor error al utilizar un mapa con mayor resolución (5 cm.) contra el método por correlación que presentó un error de 30 cm. en la posición (mas no en la orientación), sin embargo el tiempo utilizado por la búsqueda de discontinuidades (5 min.) supera el utilizado por la correlación borrosa de imágenes (10 s).

Tabla 8.1: Comparación de métodos de creación del mapa y auto localización.

8.5.1 Movimiento Autónomo

La red neuronal del T×8 que controla la navegación se comportó satisfactoriamente bien, no fue necesario reentrenarla para ser sensible a las láminas de cartoncillo, utilizadas para recubrir las mesas del ambiente de trabajo y no colisionó contra las mesas; únicamente tuvo problemas con una columna revestida de hule espuma, ya que nunca fue entrenada con dicho material. En cuanto al método del R2 para seguir la tuta más lejana, éste resultó demasiado simple e hizo que en algunas ocasiones el robot realizara movimientos erróneos debido a pequeñas separaciones entre las láminas de cartoncillo, necesarias para realizar una localización correcta por búsqueda de discontinuidades.

Por otro lado el tiempo requerido para realizar el mapa en tiempo real por el T×8 con los métodos descritos en el presente trabajo (10 min. incluyendo el recorrido) resultó mucho menor que el utilizado por el R2 (40 min. de recorrido más 10 min. de procesamiento fuera de línea).

8.5.2 Creación del Mapa Métrico

Como puede apreciarse en la Tabla 8.1, el mapa creado por el R2 (de 200×160 nodos) posee una mayor resolución comparado con el del T×8 (34×27 nodos) dado que las dimensiones de las celdas del mapa métrico del R2 son mas pequeñas que las definidas para el T×8, sin embargo el procesamiento de las observaciones obtenidas por el R2 para crear el mapa métrico, debe hacerse fuera de línea y tarda mucho mas tiempo (10 min.) que el método utilizado por el T×8 que es inmediato, ya que el mapa se genera y se complementa con cada movimiento sucesivo del robot.

Por otro lado el barrido efectuado en cada parada por el R2, realmente hace lento el procedimiento de generación del mapa, ya que en cada paso, dicho robot debe girar los 360 grados para realizar una lectura completa).

8.5.3 Creación del Mapa Topológico

El R2 no elabora ningún tipo de mapa topológico. En el caso del T×8 el mapa topológico se crea fuera de línea en aproximadamente 30 s, una vez que se ha indicado al robot que detenga el proceso de toma de muestras y navegación autónoma.

8.5.4 Creación de la Red de Conectividad

En este caso como el R2 no calcula un mapa de conectividad entre los nodos fuera de línea, sino que en cada movimiento éste debe evaluar todas las posibilidades de movimiento antes de tomar una decisión. El algoritmo de Dijkstra implementado en el T×8 proporciona casi instantáneamente la mejor ruta entre los puntos origen y destino, mientras que el método implementado en el R2 debe esperar entre 30 y 60 s antes de determinar su próximo movimiento.

8.5.5 Auto localización

El método de auto localización por correlación de imágenes es mucho más rápido que el de búsqueda de discontinuidades en las superficies de los objetos, sin embargo proporciona un mayor error de localización, ya que localizó al T×8 en un nodo vecino del nodo real.

Dado que la distancia entre nodos es grande (30 cm. en el caso del T×8), dicho error de localización es suficiente para llevar al robot a colisionar con una mesa al tratar de rodearla. No obstante el movimiento por campos potenciales lo alejaba de la mesa, la ruta lo llevó de nuevo hacia la misma, hasta colisionar con ella.

La localización por búsqueda de discontinuidades utilizada en el R2 lo ubicó correctamente con un error de 5 cm. y no le afectó significativamente para rodear una sección de mesas, sin embargo se requirieron casi 2 minutos de procesamiento para que el R2 encontrara su posición.

8.5.6 Resumen

Los métodos implementados en el T×8 mediante redes neuronales artificiales para la navegación autónoma y creación del mapa, resultaron mejores que los algoritmos determinísticos utilizados en el R2, ya que los primeros implican muy poco tiempo de procesamiento fuera de línea y permiten crear el mapa métrico en tiempo real.

La red neuronal de movimiento autónomo del T×8 operó sin ningún tipo de ajuste y, con un poco más de casos de entrenamiento, podría llegar a seguir sin problemas la superficie de prácticamente cualquier material. La utilización de campos potenciales del T×8 ayudó a evitar fácilmente la colisión contra el robot R2, sin embargo como es sabido, no resulta muy útil cuando el objeto se encuentra bloqueando perpendicularmente la ruta. En este punto bastaría con sacar el nodo bloqueado por el obstáculo, de la lista de nodos navegables y encontrar una nueva ruta. Sin embargo, además de tener que esperar algunos segundos antes de obtener dicha nueva ruta, se debería volver a marcar el nodo como navegable una vez se supere el obstáculo, de lo contrario dicho nodo quedaría bloqueado permanentemente.

Para el caso de auto localización sería recomendable disminuir las dimensiones de las celdas de la malla del T×8, fijadas originalmente en 30×30 cm., sin embargo se tendrá un aumento considerable de los tiempos empleados para generar la red de conectividad y realizar la correlación de imágenes. Lo anterior tiene la finalidad de observar si aumentando la resolución de las imágenes a operar en la correlación, se logra un menor error de localización.

Finalmente considerando las capacidades de movimiento y reacción que se pretende que posean los robots de servicio, los métodos descritos en el presente trabajo constituyen una alternativa viable para dotar de capacidades autónomas de percepción, movimiento y una toma rápida de decisiones a los robots móviles.

Capítulo 9

Conclusiones

9.1 Resumen

El presente trabajo tuvo la finalidad de poner a prueba las capacidades de aprendizaje de los modelos basados en redes neuronales artificiales. Para poder comparar los resultados de la aplicación de tales modelos, inicialmente se utilizó un ambiente virtual controlado para implementar algoritmos de movimiento, creación del mapa y localización, que no involucrasen técnicas de aprendizaje.

Para el movimiento autónomo del robot se programó un algoritmo de seguimiento de paredes, mientras que para la creación del mapa métrico y topológico se utilizaron las lecturas realizadas por el robot virtual durante el seguimiento de paredes para generar una representación métrica del ambiente en forma de malla, en donde cada medición realizada por los sonares virtuales era utilizada para marcar celdas (nodos) como navegables o bloqueados. En cuanto a la localización se intentó directamente alimentar las mediciones en bruto a una red neuronal y tratar de entrenarla para que proporcionara como salida el número de nodo y orientación correspondiente a la medición efectuada.

En cuanto a los métodos anteriores se concluye lo siguiente:

1.- No existió problema alguno en implementar y probar los algoritmos de seguimiento de paredes, únicamente se debe tomar en cuenta que en ocasiones es posible que el robot quede girando indefinidamente hacia el mismo lado (por ejemplo hacia la izquierda) cuando no se encuentra alineado con las paredes y llega a la esquina o vértice de una pared u objeto. Lo anterior se resuelve evitando que el robot pueda girar varias veces en la misma dirección previniendo que se quede atascado en una ubicación.

2.- En cuanto a la creación del mapa métrico y topológico, resultó más conveniente partir de un mapa completamente ocupado y con cada medición ir marcando como navegables las celdas con un método de “barrido” que partir de un mapa completamente vacío e ir marcando celdas como ocupadas.

Lo primero garantiza que al interior de los objetos grandes no existan celdas libres que puedan ser utilizadas para el cálculo de alguna ruta de navegación, no obstante en ocasiones puede marcar como navegables las esquinas de objetos, dependiendo del método utilizado para "limpiar" las celdas.

3.- Con suficientes épocas de entrenamiento es posible hacer que una red neuronal, en principio, proporcione en función de las mediciones en bruto de los sonares, el número de nodo y orientación del robot correspondiente a cada medición, siempre y cuando las mediciones alimentadas a la misma sean exactamente iguales a aquellas con las que fue entrenada.

Probados los métodos anteriores se procedió a incorporar ruido en el ambiente virtual tanto a las mediciones de los sonares como a los movimientos del robot con los siguientes resultados:

1.- El seguimiento de paredes no se vio afectado para errores absolutos en las mediciones de hasta 50 cm., a partir de este punto el problema mayor lo constituyen las zonas relativamente estrechas ya que las lecturas erróneas evitan que el robot recorra toda la superficie navegable.

2.- Como resultado de lo anterior, los errores en las mediciones mayores a 50 cm. provocan principalmente que el mapa quede incompleto. Errores menores a dicho valor prácticamente no alteran la creación del mapa.

3.- Cuando se alimenta la red neuronal con un vector ligeramente distinto a aquellos entrenados, proporciona una ubicación y orientación completamente errónea. Lo cual se puede atribuir a que, para poder predecir correctamente la el nodo y orientación a partir de las mediciones de los sonares, la red se ajusta casi perfectamente a los datos de entrenamiento, es decir, existe un sobreajuste, lo cual hace imposible que pueda predecir con certeza la posición del robot cuando existen ligeras variaciones en los datos de entrada.

4.- En cuanto a los errores en los movimientos del robot se observó que los métodos de navegación y creación del mapa no resultaron tan sensibles a errores de translación de hasta 10 cm., sin embargo bastó con un error de 5 grados en la rotación para que, luego de varios movimientos, el mapa generado sea completamente erróneo.

Una vez que se evaluó el desempeño de los algoritmos anteriores en la presencia de errores de medición y movimiento, se calibró al robot real para evitar llegar a los niveles de error de movimiento críticos (10 cm. en translación y 5 grados en la rotación). Cabe señalar que los movimientos del T×8 son bastante precisos luego de ser calibrado, con un error máximo de 1 cm. en la translación y 2 grados para la rotación. Con el robot calibrado se procedió a probar los métodos desarrollados. Es posible concluir lo siguiente:

1.- El método de seguimiento de paredes fracasó rotundamente ya que no pudo hacer que el robot avanzara por más de 2 m sin chocar contra algún objeto.

2.- Debido a que el robot no pudo recorrer debidamente el ambiente de trabajo no fue posible evaluar la creación del mapa y localización.

En este punto se decidió evaluar el desempeño de los sonares del T×8 para observar si el error en las mediciones presentaba una distribución gaussiana como la supuesta en el ambiente virtual y superaba los 50 cm. Se evaluaron distintos tipos de materiales y distintos ángulos de incidencia de la señal del sonar sobre la superficie del material y se encontró lo siguiente:

1.- Cuando la señal del sonar incide perpendicularmente a la superficie del objeto, sin importar si se trata de tela, cartón o metal, la medición del sonar es realmente precisa (menor a 1 cm. de error) para todos los sonares del T×8 .

2.- Dependiendo del material, cuando se supera cierto ángulo máximo de variación con respecto a la perpendicular (ver sección 5.10.2), el sonar comienza a proporcional lecturas erróneas y comúnmente en forma cíclica, siempre por encima del valor real y en aumento (p. ej. si el objeto se halla a 20 cm. el sonar proporciona 30.5, 42.5, 60.7, 70.6, 30.5, 42.5 ...). Mientras mayor sea dicha variación del ángulo de incidencia, mayor es el error de lectura del sonar, con lo cual se observó que el error de medición de los sonares comúnmente rebasa los 50 cm. y no posee de ninguna manera una distribución gaussiana.

Inmediatamente se procedió a evaluar las capacidades de las redes neuronales y entrenar una red para realizar la tarea del seguimiento de paredes. Dicha red debía recibir como entrada las mediciones de los sonares del robot con algún tipo de preprocesamiento que le facilitase el aprendizaje, y proporcionar como salida la dirección del siguiente movimiento del robot. Las conclusiones son las siguientes:

1.- La red implementada de tipo *feed-forward* con una ecualización del histograma como preprocesamiento a los datos funcionó realmente bien con relativamente pocos casos de entrenamiento y diferentes materiales.

2.- El método de obtención del conjunto de entrenamiento utilizando una palanca de juegos (joystick) facilitó enormemente la tarea.

3.- Para la obtención del conjunto final de pesos de la red, se tuvo que evaluar el desempeño de cada red entrenada y aumentar los casos de entrenamiento con algún tipo de material al que la red se mostrara poco sensible. Lo anterior significó detener el entrenamiento evaluando directamente el desempeño de la red en la práctica siguiendo las paredes y no solo cuando se presentase el menor error en el conjunto de prueba, lo cual implica que no siempre detener el entrenamiento cuando se presenta el menor error en el conjunto de prueba lleve a los mejores resultados en la práctica.

Con el movimiento autónomo utilizando la red neuronal funcionando correctamente fue posible evaluar los algoritmos utilizados en el ambiente virtual para la creación del mapa y la localización con las siguientes conclusiones:

1.- Debido a los grandes errores en las mediciones de los sonares explicadas anteriormente si se parte de un mapa completamente lleno y se va "rasurando" en función de las lecturas, el mapa final resultó casi vacío, mientras que si se parte del caso contrario donde se posee un mapa vació y se marcan las celdas o nodos como no-navegables, resultó un mapa prácticamente lleno o imposible de navegar.

2.- Similarmente por tales errores en los sonares, la red entrenada para la auto localización con los datos del ambiente virtual fracasó rotundamente. Cabe aclarar que no fue posible reentrenar dicha red con los datos del ambiente real, lo que suponía tomar varias lecturas con diferentes orientaciones parta cada nodo navegable del mapa real y, sabiendo de entrada la susceptibilidad de la red al más mínimo error de lectura, supondría un tiempo realmente considerable y prácticamente inútil para los objetivos del presente trabajo que, si bien toca el tema de la localización, no representa el objetivo principal del mismo.

Una vez observadas las deficiencias de los métodos determinísticos tanto para la navegación autónoma como para la creación del mapa se implementó una Red Neuronal de Agrupamiento o Clustering, la cual tendría la función de, luego de llevar las mediciones de los sonares directamente de un vector de 16 dimensiones a un plano *xy* donde cada medición es ubicada como un punto, buscar las agrupaciones de dichos puntos para categorizar a cada grupo o cluster como un objeto en particular, representando un conjunto de mediciones como un objeto de dimensiones fijas localizado en el centroide del cluster de puntos agrupados. Se obtuvo lo siguiente:

1.- La red de agrupamiento permite trabajar en tiempo real, es decir, conforme se generan nuevas mediciones éstas son utilizadas para entrenar a la red.

2.- Resulta fundamental determinar el número inicial de grupos o clusters (igual al número de neuronas de la red). Una alternativa es definir una malla de neuronas cubriendo cierta área alrededor del punto de partida del robot, que se supone como (0.0) viendo a 90 grados, sin embargo se propone un método basado en creación de grupos o clusters sobre demanda que inicia con un solo cluster y, conforme se necesiten, se generan nuevos clusters.

3.- Al realizar un mapeo de cada medición del sonar a un punto del plano *xy*, también las lecturas erróneas son mapeadas como puntos en el plano. Al tomar en cuenta la red de agrupamiento todos los puntos por igual, el mapa resultante no difiere mucho de aquel generado determinísticamente bloqueando las celdas o nodos correspondientes a cada medición, es decir, se obtiene un mapa prácticamente lleno. La ventaja es que, con la red de agrupamiento, se conoce la ubicación exacta de cada grupo o cluster, a diferencia del método determinístico.

A continuación se introdujo el concepto de Red Borrosa de Agrupamiento, agregando capacidades borrosas a la red de agrupamiento, de tal suerte que los puntos a agrupar no se tomasen por igual. Se determinó un grado de pertenencia o "confiabilidad" de cada punto en el plano, en función de la magnitud de la medición que lo originó, es decir, mientras más lejano estaba el objeto detectado por el sonar del robot, menos confiable resulta tal medición y viceversa. Lo anterior automáticamente hará que, en el caso de presentarse mediciones erróneas cíclicas y crecientes, aquellas de menor magnitud (y con ello más cercanas al valor real) serán tomadas en cuenta en mucho mayor grado que el resto de las mediciones. De esto se puede concluir que:

1.- La adición del factor borroso de operación a la red de agrupamiento permitió no solo filtrar una gran cantidad de mediciones erróneas, sino que sentó un precedente para poder agregar otros factores tales como confiabilidad en particular de cada sonar, confiabilidad de la medición respecto al tiempo o a la exactitud de la localización del robot, etc.

2.- El mapa métrico obtenido resultó bastante cercano al mapa real, sin embargo es necesario definir a priori las dimensiones de cada grupo o cluster para que los objetos encontrados no sean demasiado grandes como para bloquear posibles rutas, ni demasiado pequeños como para hacer lentos los cálculos de conectividad.

3.- Tal tipo de red puede ser utilizada a diferentes escalas, es decir, una vez encontradas las celdas navegables, se puede aplicar nuevamente la red borrosa para encontrar los grupos o clusters de celdas navegables que puedan representar zonas o nodos a una escala mayor.

Para la localización se evaluó el aprendizaje en una red autoasociativa, tratando de hacer que, por cada nodo, una red entrenada con las distintas vistas locales para dicho nodo pudiera indicar si las observaciones realizadas por el robot en la ubicación desconocida, correspondían con alguna de las vistas locales almacenadas por la red autoasociativa, eligiendo como posible candidata a la red o nodo con la mayor función de energía. Es posible concluir lo siguiente:

1.- Los patrones de aprendizaje (vistas locales) correspondientes a diferentes orientaciones del robot para un nodo en particular resultan bastante similares para la red (a pesar de existir una rotación entre ellas), por tal motivo no es posible reducir el error al entrenarla, lo que significa que le resulta imposible poder diferenciarlas (lo que no ocurre cuando se entrena con patrones visuales correspondientes a los dígitos del 0 al 9).

2.- Por el motivo anterior la red de Hopfield no constituye una alternativa para el problema de la localización utilizando patrones correspondientes a vistas locales.

Una vez evaluados los problemas de este tipo de red se comparó su desempeño contra un método basado en correlación de imágenes, en donde cada vista local generada a

partir de las observaciones locales del robot, servía para generar un filtro o kernel que, luego de aplicarse a una representación binaria del mapa del entorno, proporcionara la ubicación más aproximada a la real. En este punto se hizo uso nuevamente de conceptos de lógica borrosa para encontrar la ubicación de máxima coincidencia, en función de distintas hipótesis de localización. Luego de analizar los resultados obtenidos es posible decir que:

1.- La operación conocida como "suma de Hammacher" (cap. IV, tabla 4.1) permite "fusionar" las distintas hipótesis de localización basados en la premisa de que, para la ubicación real, se encuentra una alta correlación entre el kernel y el mapa en las orientaciones cercanas a la real. Dicha operación permite ignorar ubicaciones erróneas con un mayor grado de correlación para una sola imagen en particular, pero casi nula en el resto.

2.- La desventaja del método consiste en que es necesario determinar a priori el tamaño del kernel o vista local; si éste resulta demasiado pequeño y existe una separación mucho mayor entre los objetos, pueden generarse vistas locales vacías que generen ubicaciones erróneas. Por otro lado si la vista local es demasiado grande el método de correlación tendrá problemas para encontrar ubicaciones cercanas a los bordes del mapa.

En cuanto a los objetivos planteados es posible concluir lo siguiente:

a) Desarrollar un método basado en redes neuronales que permita a un robot móvil:

1) Navegar libremente por el ambiente de trabajo.

El objetivo fue alcanzado, ya que fue posible encontrar la arquitectura correcta (red progresiva con normalización de los datos de entrada) que le permitiese al robot navegar libremente, utilizando las lecturas de sus sonares. Se probó su funcionamiento para diferentes ambientes de operación y se determinó experimentalmente que se requiere efectuar entrenamientos con distintos tipo de materiales, incluyendo hule espuma, para garantizar su eficiencia.

2) Elaborar una representación de su entorno, con base en las lecturas proporcionadas por sus sonares durante el recorrido, tomando en cuenta los problemas que se pueden presentar con tales mediciones.

Este objetivo fue cubierto. Se desarrolló y probó una Red de Agrupamiento Borrosa, que localiza los objetos del medio ambiente como una serie de objetos de dimensiones predefinidas, correspondientes a los grupos o *clusters* encontrados por la red de agrupamiento.

b) Evaluar el desempeño de dicho método con un robot y ambiente real.

Se evaluó el desempeño de ambos métodos con el robot T×8 y se comparó su eficiencia contra métodos no neuronales, en el ambiente real de operación. Se obtuvieron mejores resultados utilizando redes neuronales artificiales (excepto en la auto localización).

De manera general y evaluando los resultados obtenidos es posible concluir:

El uso de redes neuronales para las tareas de navegación autónoma y creación del mapa métrico mediante búsqueda de grupos o clusters resulta apropiado para los robots móviles, en donde se requiere rapidez y operación en tiempo real, además permiten que se invierta tiempo en entrenar tales redes, en lugar de programar y ajustar algoritmos determinísticos para diferentes condiciones de operación.

La adición de factores borrosos a las redes de agrupamiento facilita enormemente las tareas de filtrado de mediciones erróneas y abre la posibilidad de contemplar distintos factores que intervienen en su correcta operación, tales como confiabilidad de la medición o grado de certeza de la posición del robot.

En cuanto a la localización, no obstante las redes *feed-forward* pueden ajustarse perfectamente a los datos de entrenamiento y predecir el nodo y orientación en función exclusivamente de las mediciones de los sonares, no se recomienda su uso alimentando directamente las mediciones de entrada ya que cualquier variación por mínima que sea dará como resultado una localización errónea del robot.

Las redes autoasociativas, como la red de Hopfield implementada en las tareas de localización con vistas locales, no resultan convenientes ya que, para que tales redes puedan diferenciar los patrones, éstos deben ser suficientemente distintos entre sí para facilitar la formación de las llamadas "cuencas de atracción", lo cual no ocurre cuando se toman como patrones una misma vista local girada en distintos ángulos.

9.2 Aportaciones

El presente trabajo realiza importantes aportaciones en la resolución del problema SLAM, ya que evalúa directamente el desempeño de las redes neuronales artificiales en las tareas de navegación autónoma y creación del mapa métrico:

- Confirma que el uso de redes neuronales artificiales para las tareas de movimiento autónomo en los robots móviles es altamente recomendable, dada su capacidad de aprendizaje y por la relativa simplicidad del proceso de entrenamiento.

- Presenta una alternativa fácil de desarrollar y de probar para que los robots móviles realicen recorridos por ambientes desconocidos, utilizando la información proporcionada por sensores de tipo sonar, pero siendo extensible a sensores de tipo láser o infrarrojos.

- Se comprueba la capacidad de respuesta casi inmediata de tales redes, lo que permite a los robots móviles reaccionar rápidamente a situaciones inesperadas.

- Una vez definida la arquitectura y entrenada la red neuronal de movimiento autónomo, ésta puede implementarse en la práctica mediante operaciones con matrices y operadores de activación (como las ecuaciones 3.7 y 3.8). Lo anterior

permite que la red de movimiento autónomo pueda implantarse en dispositivos con limitada capacidad de procesamiento y almacenamiento.

La aportación más importante al estado del arte, es el concepto *Red Borrosa de Agrupamiento*, donde el término *borroso* no se aplica a la salida de la red como en otros casos [Höppner 99], sino la característica borrosa de los datos de entrada de la propia red:

- Se marca un criterio para el entrenamiento de la red, donde no se toma con igual validez a todos los vectores de entrenamiento, por el contrario, para cada vector de aprendizaje se establece un grado de certidumbre o *confiabilidad*. La red neuronal borrosa ajusta sus pesos en el sentido de las lecturas con mayor *confiabilidad*.

- El procedimiento anterior se aplicó para una red de agrupamiento o *clustering*, sin embargo es extensible a redes *feed forward* o inclusive redes auto asociativas (entrenadas utilizando el método de Greville [Kohonen 89]), donde se actualice el vector de pesos de cada neurona en una fracción (ritmo de aprendizaje) del vector de entrenamiento alimentado a la red, es decir, donde se tenga una regla de actualización de la forma:

$$\overline{w}_k(t+1) = \overline{w}_k(t) + \gamma \; f(\overline{x}(t), \overline{w}_k(t))$$

donde $\overline{w}_k$ es el vector de pesos de la neurona *k*, γ es el ritmo de aprendizaje y $\overline{x}$ es el vector de entrenamiento.

9.3 Comparación vs. Métodos No Neuronales

Desarrollar métodos determinísticos para calcular los movimientos del robot no resultó efectivo en robots móviles con sensores que proporcionen un error que no pueda ser caracterizado con facilidad (como en el caso de los sonares) ya que resulta prácticamente imposible prever todas las situaciones que arrojen posibles lecturas erróneas y confundan al algoritmo de movimiento.

Cuando se cuenta con sensores que no presentan tales errores (como el láser) un algoritmo determinístico siempre tendrá problemas al enfrentarse a situaciones inesperadas, dada su nula adaptabilidad a condiciones no previstas, además se tiene el problema inherente de determinar con precisión el siguiente paso del robot en función de las lecturas obtenidas. Cuando existan lecturas de los sensores apenas por encima o por debajo de cualquier límite impuesto, el robot podrá desplazarse en direcciones completamente distintas o inclusive erróneas (como el robot R2 en la sección de resultados)..

El uso de redes neuronales artificiales permitió una menor sensibilidad a variaciones de los datos de entrada (correspondientes a errores en las mediciones de los sonares), sin embargo presenta el problema de que es necesario entrenar a la red con *suficientes* datos de entrenamiento, siendo la dificultad principal determinar cuántos y cuáles casos son *suficientes* para la red.

En cuanto a la capacidad de respuesta las redes neuronales artificiales, éstas permiten una toma de decisión casi instantánea y pueden trabajar en tiempo real sin dificultad para tareas de movimiento autónomo y agrupamiento o clustering.

Los principales problemas que presentan las redes neuronales en cuanto al movimiento autónomo son

a) Establecer la arquitectura correcta y
b) Una vez entrenada la red, esta sólo funcionará en el robot para el cual fue diseñada.

La incapacidad de las redes neuronales de adaptarse a nuevas configuraciones una vez entrenadas (por ejemplo un robot con otro método de locomoción o diferente número de sonares) radica en su operación de tipo "caja negra", en donde se conocen los parámetros exactos (pesos) de su función de operación, pero resulta casi imposible modificarla determinísticamente para adaptarse a nuevas configuraciones. No obstante lo anterior se recomienda el uso de dichas redes para obtener resultados en un corto plazo.

En cuanto a la localización los métodos no neuronales tuvieron mejor desempeño, sin embargo requieren más tiempo para obtener la posición que las redes neuronales artificiales. La red *feed forward* se ajusta demasiado a los datos de entrada y se hace muy sensible al ruido, mientras que la red de Hopfield se ajusta muy poco al tratarse de patrones similares. Es necesario probar con algún tipo de red realimentada que refine la posición con sucesivas lecturas e implemente localización débil (ver sección 2.6).

9.4 Redes Neuronales vs. Redes Neuronales Borrosas

La adición de valores borrosos a los datos de entrada de las redes neuronales artificiales permitió mejorar su funcionamiento para el caso de la red de agrupamiento, ya que encontró clusters más próximos a los objetos del ambiente y no dio importancia a una gran cantidad de lecturas erróneas.

Al tomar por igual todos los vectores de entrenamiento una red no borrosa inmediatamente toma como válidas tanto las lecturas correctas de los sonares como las incorrectas. Si bien no es posible *a priori* determinar cuáles lo son y cuales no lo son, la lógica borrosa permite determinar de forma aproximada un conjunto de mediciones válidas, donde el valor de la función de pertenencia de cada medición sirve como parámetro de ajuste adicional al ritmo aprendizaje.

Si se combinan varias características borrosas (como certeza en la ubicación del robot o el tiempo transcurrido desde que se tomó la medición) dentro de un solo valor borroso o *confiabilidad*, es posible dotar a la red de cierto grado de adaptabilidad a las condiciones del medio, modificando sus vectores de pesos hacia las lecturas con mayor grado de certeza o importancia conforme transcurra el tiempo. Inclusive puede llegar a ignorar mediciones (y con ello eliminando clusters u objetos en el mapa métrico) en cuanto disminuya la *confiabilidad* de dichas mediciones. Lo anterior permitiría a la red de

agrupamiento contar con un mapa dinámico donde los objetos en movimiento puedan aparecer y luego desaparecer.

El principal problema con las redes borrosas lo constituye la determinación de la función de pertenencia óptima que indique el grado de *confiabilidad* de cada medición. Si bien en nuestro caso resultó relativamente sencillo, en la mayoría de los casos se cuenta con los datos de entrenamiento pero se desconocen totalmente las condiciones en que fueron tomadas las muestras.

Sin embargo, como en el caso del entrenamiento del movimiento del T×8 con el joystick, cuando es posible determinar al momento de generar los datos de entrenamiento qué tan bueno resultó dicho entrenamiento (en función de lo que se espera que el robot haga) es posible *a priori* asignar a todos los datos generados durante cada sesión de entrenamiento un valor de *confiabilidad* entre 0 y 1 que sirva como entrada de una red borrosa, ya sea de agrupamiento o de tipo progresivo. Una vez que se haya entrenado la red, si al momento de tomar la lectura no es posible calcular el grado de *confiabilidad* de la medición se podrá asignar la máxima confiabilidad a cada una, no obstante la red habrá "aprendido" tomando los mejores ejemplos de entrenamiento.

9.3 Trabajos Futuros

El primer problema que se propone resolver consiste en la mejorar la auto localización utilizando la red neuronal de tipo *feed-forward* ya que probó ser capaz de "aprender" a localizarse tan solo con las mediciones de los sonares. Se recomienda probar con algún tipo de preprocesamiento que la haga menos sensible a pequeñas variaciones en los datos de entrada, como puede ser la cuantificación vectorial [Brío 02, pp. 158].

Se propone utilizar la localización por correlación de imágenes exclusivamente para determinar la zona o "habitación" del mapa en la que se encuentra el robot, una vez que se han encontrado las zonas del mapa, ya que su operación es bastante rápida pero requiere fijar de antemano parámetros que pueden hacer que opere de manera errónea si no se fijan adecuadamente. Se propone trabajar con algún tipo de red realimentada en donde la hipótesis de localización pueda ser modificada a medida que el robot avanza y se generan nuevas observaciones.

Será necesario agregar al entrenamiento de la red neuronal de seguimiento de paredes, algunos casos de entrenamiento que involucren materiales absorbentes de la señal del sonar y observar si, en términos generales, el desempeño de tal red mejora. Por otro lado, ya que no es posible garantizar que en todos los casos el seguimiento de paredes logre recorrer todo el espacio de trabajo, se propone entrenar otras redes *feed-forward* con distintas formas de recorrer el ambiente, por ejemplo buscando la distancia máxima. Resultaría interesante hacer que el robot utilice una u otra red dependiendo de cada situación en particular.

Referencias

[Amigoni 03] Amigoni, F., Caglioti, V., *An information-based criterion for efficient robot map building.* In Proceedings of the IEEE International Symposium on Virtual Environments, Human-Computer Interfaces and Measurement Systems (VECIMS2003), páginas 184-189.

[Amit 86] Amit, D. J., Gutfreund, H., & Sompolinsky, H. (1986). *Spin-glass models of neural networks*. Physical Review A., 32 (2), pp. 1007-1018.

[Arkin 98] Arkin, R., *Behavior-based Robotics*, MIT Press, Cambridge 1998

[Brío 02] Bonifacio Martín del Brío, Alfredo Sanz Molina, *Redes Neuronales y Sistemas Borrosos*, 2ª Edición, Alfaomega 2002 pp. 129-132

[Curtis 87] Curtis, T.E. & Wickenden, J.T. *Signal Processing for Sonar: A Brief Review of the Algorithms, Architectures and Enabling Technology*, IERE 5th int. Conf. on Electronics for Ocean Technology, Marzo 1987, Heriot-Watt University.

[Dijkstra 59] Dijkstra, E., *A Note on Two Problems in Connection with Graphs*, Numeriske Matematik, (1): 269-271, 1959

[Dudek 00] Dudek, G., Jenkin, M. *Computational Principles of Mobile Robotics*. Cambridge , University Press, 2000

[Flake 98] G.W. Flake, *Square unit augmented, radially extended, multilayer perceptons*, Neural Networks: Tricks of the Trade (G. B. Orr & K.-R. Muller, eds.), vol. 1524 de Lecture Notes in Computer Science, pp. 145-163, Berlin: Spinger Verlag, 1998.

[Gers 00] F.A. Gers, J. Schmidhuber, and F. Cummins, *Learning to forget: Continual prediction with LSTM*, Neural Computation, vol. 12, no. 10, pp. 2451-2471, 2000.

[Gers 01] F.A. Gers and J. Schmidhuber, *Long Short-Term Memory learns simple context free and context sensitive languages*, IEEE Transactions on Neural Networks, 2001.

[González 92] Gonzalez, R.C., Woods, R.E., *Digital Image Processing*. Addison-Wesley, Estados Unidos 1992

[Hertz 91] Hertz, J., Krogh, A., Palmer, R.. *Introduction to the theory of Neural Computation*. Addison-Wesley 1991.

[Hopfield 82] Hopfield, J.J. *Neural networks and physical systems with emergent collective computational abilities*. Proc. of the National Academy of Sciences, 79, pp. 2554-2558, 1982.

[Hochreiter 97] S. Hochreiter and J. Schmidhuber, *Long short-term memory*, Neural Computation, vol. 9, no. 8, pp. 1735-1780, 1997.

[Höppner 99] F. Höppner, F. Klawonn, R. Kruse, T. Runkler: *Fuzzy Cluster Analysis*. Wiley, Chichester, 1999.

[Klapper 01] M. Klapper-Rybicka, N. N. Schraudolph, J. Schmidhuber. *Unsupervised Learning in LSTM Recurrent Neural Networks*, G. Dorffner, H. Bischof, K. Hornik, eds., Proceedings of Int. Conf. on Artificial Neural Networks ICANN'01, Viena, LNCS 2130, pp. 684-691, 2001.

[Kohonen 89] Kohonen T., *Self-Organization and Associative Memory*. 3th edition, Springer-Verlag, 1989.

[Krose 96] Krose, B., van der Smagt, P., *An introduction to neural networks*, University of Amsterdam, Octava edición, Noviembre de 1996

[Kuri 02] Kuri, A., *Automatic Clustering with Self-Organizing Maps and Genetic Algorithms*, 3rd WSEAS Conference on Neural Networks and Applications, Interlaken, Suiza, pp. 173-180, February, 2002.

[Latombe 91] Latombe, J. C., *Robot Motion Planning*, Kluwe Academic Publisher, 1991

[McCulloch 43] McCulloch, W.S. Pitts, W. *A logical calculus of the ideas immanent in nervous activity*. Bulletin of Mathematical Biophysics, 5, 115-133, 1943.

[Minsky 69] Minsky, M.., Papert, S. *Perceptrons: An Introduction to Computational Geometry*. MIT Press, 1969.

[Pawson 86] Pawson, R., *El Libro del Robot*, Ed. Gustavo Gili, Barcelona 1986

[Personnaz 86] Personnaz L., Guyon, I., Dreyfus, G. *Collective computational properties of neural networks: New learning mechanisms*. Physical Review A. 34, 5, pp. 4217-4428, 1986

[Poggio 90] Poggio. T., Girosi, F. *Networks for approximation and learning*, Proc. of the IEEE, Sept., 1481-1497, 1990.

[Rumelhart 86] Rumelhart, D. E., Hinton G. E., Williams R.J. *Learning representations by backpropagating errors*. Nature, 323, 533-6, 1986

[Rosenblatt 62] Rosenblatt, F. *Principles of Neurodynamics*. Spartan Books, New York, 1962.

[Savage 98] Savage, J. et al., *The ViRbot: a virtual reality mobile robot driven with multimodal commands*, Expert Systems with Applications. 15: 413-419

[Schank 72] Schank , R. *Conceptual Dependency: a theory of natural language understamding*. Cognituve Psychology 3, 4 (Octubre 1972), 552-631

[Schraft 00] Schraft , R.D., Schmierer, G. *Service Robots*. A. K. Peters, 2000

[Thrun 03] Thrun S., Montemerlo M.., Koller D., Wegbreit B., *FastSLAM: An Efficient Solution to the Simultaneous Localization And Mapping Problem with Unknown Data Association*, Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), 2003.

[Widrow 60] Widrow, B., Hoff, M..E. *Adaptive switching circuits*. 1960 IRE WESCON Convention Record, 4, pp. 96-104, 1960

[Zadeh 65] Zadeh, L. A. *Fuzzy Sets*. Information & Control, 8, 338-353,1965

[Zhidong 97] Zhidong L., Zhang B., *A Fuzzy Elman Neural Network*, The State Key Lab of Intelligent Technology and Systems Dept. of Computer Science, Tsinghua University Beijing 100084, China, 1997, http://www.cs.caltech.edu/~ling/pub/97fenn.pdf

[Zimmermann 90] Zimmermann, H. J., *Fuzzy Theory and Its Application*, Kluwer Academic Publishers, Boston, Massachussets, 1990, pp. 32

Glosario

El presente trabajo aborda diversos conceptos descritos y planteados por sus autores inicialmente en idioma inglés, cuya traducción literal al castellano puede resultar incorrecta o vaga en términos del sentido original que posee cada concepto. A continuación se especifican los términos o conceptos cuya traducción literal pudiera resultar confusa y que se incluyen dentro del trabajo sin realizar traducción alguna.

Lógica Crisp : Lógica que sólo acepta valores verdaderos o falsos.

Fuzzy Logic : Lógica "borrosa" o "difusa". El valor de las proposiciones o variables lógicas puede ir desde un valor mínimo hasta un valor máximo.

Fuzzyfier : Método utilizado para convertir valores *crisp* en valores borrosos.

Defuzzyfier: Método utilizado para convertir valores borrosos en valores *crisp*.

Feed-forward: Referente a las redes neuronales progresivas, donde las unidades o neuronas actualizan su valor desde las capas de entrada, hacia las capas de salida. Ninguna unidad posee realimentación consigo misma ni con las demás unidades de su capa. Tampoco conecta sus salidas con alguna(s) unidad de una capa inferior (más próxima a las entradas).

Feedback: También se le llama retroalimentación, se presenta cuando la salida de alguna neurona se conecta a la entrada de sí misma, o a la entrada de alguna(s) neurona de una capa inferior (más próxima a las entradas).

Cluster: Conjunto de puntos o vectores en torno a un núcleo o centroide, que para efectos de cálculo son tomados como una unidad de dimensiones fijas.

Clustering: Proceso mediante el cual se encuentran los grupos o *clusters* correspondientes a un conjunto de vectores o puntos en el plano.

Adaline: Proviene de Adaptive Linear Element, uno de los primeros modelos de neurona artificial.

Dead Reckoning: Estimación de la posición utilizando exclusivamente la odometría interna del robot, sin tomar en cuenta información del mundo exterior.

SLAM: Siglas de Simultaneous Localization And Mapping, problema que implica para un robot móvil moverse dentro de su entorno y al mismo tiempo estimar su posición y elaborar un mapa de su entorno.

VQ: Siglas de *Vector Quantization*. Proceso mediante el cual se representa un conjunto de patrones o vectores con cierta proximidad espacial, mediante un solo vector o prototipo de clase.