UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

FACULTAD DE INGENIERIA

DIVISION DE ESTUDIOS DE POSGRADO

Control de un robot móvil usando Redes Neuronales y Sistemas Expertos

T E S I S

QUE PARA OBTENER EL GRADO DE

MAESTRO EN INGENIERIA
(E L E C T R I C A)

P R E S E N T A :

MARCO ANTONIO MORALES AGUIRRE

DIRECTOR DE TESIS : DR. JESÚS SAVAGE CARMONA

MEXICO, D. F. OCTUBRE, 1998

A mi esposa: Aimée por todo, por el Amor, por los desvelos, por sus críticas y por ser mi permanente estímulo, pero sobre todo por su Amor. Éste es un logro de los dos.

A mi asesor: Dr. Jesús Savage Carmona que aparte de ser un verdadero maestro y amigo, siempre estuvo dispuesto a apoyarme.

A mis padres: Marcos y Esperanza que pueden ver este trabajo como un fruto más de todo lo que han sembrado.

A mi hermana: Rocío para que incremente sus metas y las logre.

Agradecimientos

Quiero agradecer el apoyo brindado por todos los que aportaron algo en el desarrollo de éste trabajo.

De manera especial le agradezco a la Facultad de Ingeniería de la Universidad Nacional Autónoma de México, por admitirme para estudiar la maestría en sus aulas.

Al Consejo Nacional de Ciencia y Tecnología le agradezco la beca crédito que me otorgó durante los dos años de estudios de maestría.

Al Dr. Jesús Savage Carmona, mi director de tesis, le agradezco permitirme trabajar con él y su invaluable asesoría.

A todos los que en el Laboratorio de Interfaces Inteligentes me dieron su apoyo también les agradezco.

Gracias a todos ellos.

Índice General

Índice de Figuras

Índice de Tablas

Resumen

El presente trabajo describe el desarrollo de un conjunto de programas para controlar un robot móvil. Se evalúa la aplicación de redes neuronales y de sistemas expertos para mezclar dos enfoques de control: el control tradicional y el control basado en comportamientos. El objetivo es contar con un robot que ejecute órdenes que indican cambios de posición y de orientación dentro de un entorno estático del cual se conoce el área ocupada por objetos distribuidos en el mismo.

El robot corresponde al modelo B14 de RWI (Real World Interface Inc.) y los distintos componentes se implantaron en la computadora a bordo del mismo. El planificador y el navegador se instrumentaron con la herramienta de desarrollo de sistemas expertos CLIPS, el piloto se codificó en la interfaz de programación del robot. Y para que el robot fuera capaz de superar obstáculos desconocidos se simuló una red neuronal y se compararon sus resultados con los obtenidos mediante un conjunto de funciones lógicas.

Se encontró que aunque la red neuronal aparentemente ofrece resultados útiles, no esradicalmente mejor que el uso de funciones booleanas. Por otro lado el sistema experto fué muy adecuado para instrumentar la planificación y navegación, ya que se le introdujeron los conocimientos necesarios para cumplir con el objetivo y siempre ofrecieron el resultado esperado.

El principal problema se presentó en el piloto pues su funcionamiento dependió siempre de la precisión del odómetro del robot y ésta no es muy adecuada. Aquí se nota que es necesario contar con un sensor de posición independiente de los actuadores.

La integración entre los bloques de control basados en comportamientos y tradicional no ofreció demasiados problemas una vez que se definió un mecanismo de comunicación estándar, por lo que se logró un sistema que permite seguir avanzando en mejoras al robot.

Capítulo 1

Introducción

1.1 Antecedentes

Un robot es un dispositivo mecánico programable que interactúa con el mundo físico para resolver problemas planteados por el hombre. Hay incluso quien dice que un robot es una conexión inteligente entre la percepción y la acción [1].

La robótica es una disciplina que emerge de la conjunción de conocimientos de Ingeniería Eléctrica, Ingeniería Mecánica, Ingeniería en Computación y Ciencias de la Computación, en la cual las fronteras entre las distintas áreas son difusas. Un robot tiene partes mecánicas que le permiten actuar. Cuenta con partes eléctricas y electrónicas que convierten señales de diversos tipos en señales eléctricas y que sirven como sensores que le permiten al robot tener conocimiento de lo que sucede en el mundo con el que interactúa. Y por último, el robot tiene un conjunto de programas, escritos en hardware, en software o en una combinación de ambos, que determinan su comportamiento.

En general, se puede decir que existen dos tipos de robots: los brazos manipuladores y los robots móviles. La diferencia entre aquellos y éstos es que los primeros tienen al menos un eslabón fijo a la tierra, mientras los segundos están libres. Como consecuencia, los robots móviles tienen más libertad para realizar toda clase de tareas que no involucran movimientos repetitivos.

En el Laboratorio de Interfaces Inteligentes (LII) del Departamento de Ingeniería en Computación de la Facultad de Ingeniería de la Universidad Nacional Autónoma de México, se ha estado realizando un esfuerzo muy grande por contar con una infraestructura adecuada para realizar investigación en robótica. Un paso en esta dirección lo conformó la adquisición de un robot móvil modelo B14 fabricado por la empresa Real World Interface. Las características del B14 permiten implantar programas rapidamente, utilizando sus sensores y actuadores con sólo hacer llamadas a funciones en lenguaje C.

Uno de los primeros proyectos que se propusieron, consiste en dotar al robot de la capacidad de obedecer a órdenes un tanto ambiguas, por ejemplo "ve hacia allá"; en donde "allá" puede ser cualquier punto dentro de una zona conocida.

Para lograr esto es necesario entender la interfaz de programación del B14, agregarle sensores y realizar los programas que permitan el comportamiento descrito. El primer problema que se debe resolver, y que es el que trata este trabajo, consiste en lograr que el robot alcanze una posición y orientación bien definidas. Ésto permitirá contar con una plataforma desarrollada para avanzar en otras direcciones.

1.2 Definición del problema

Implantar mecanismos de planificación, navegación y pilotaje en un robot móvil B14 utilizando el enfoque de control de robots tradicional y el enfoque basado en comportamientos utilizando redes neuronales para procesar información de los sensores y sistemas expertos para dotar de conocimientos al robot.

1.3 Objetivos

Hay varios objetivos en la realización de este trabajo. El principal es contar con un robot que ejecute órdenes que indiquen cambios de posición y de orientación dentro de un entorno estático del cual se conoce el área ocupada por objetos.

Para tratar de evadir objetos que no se conocían, se utilizará una red neuronal, con el fin de probarla y compararla con el desempeño de funciones booleanas. Los objetivos particulares se listan a continuación:

- Evaluar el desempeño de una red neuronal al procesar los datos que recogen los sensores de un robot móvil, y ofrecer al robot opciones de movimientos para evadir obstáculos.
- Aplicar un sistemas expertos para resolver problemas de planificación y navegación en robots móviles.
- Utilizar una combinación del enfoque de control de robots tradicional con el basado en comportamientos en un solo robot.
- Contar con un robot móvil que opere en un entorno "natural" con información incompleta del mismo, al que no se le tengan que hacer modificaciones.

1.4 Recursos

Los recursos con los que se contaron para la realización de este proyecto son:

- Un robot B14 de Real Worl Interface;
- una estación de trabajo Digital con procesador Alpha y sistema operativo DEC OSF;
- una PC con procesador Pentium y sistema operativo Linux; y,

- una red de computadoras mediante la cual se conectan todas las computadoras.

1.5 Alcances

Este proyecto tiene como propósito principal contar con un robot al que se le puedan agregar módulos de comportamiento y "conocimientos" que lo hagan mas inteligente.

Aunque los resultados se aplicarán directamente al equipo con el que se cuenta en el LII, se espera que se puedan implantar en cualquier robot que cuente con sensores y actuadores similares al del LII sin necesidad de realizar cambios importantes.

El proyecto se considera terminado en cuanto se cuente con un robot que sea capaz de llegar a un punto solicitado siguiendo una ruta óptima, generada a partir de un mapa de su entorno, y que pueda evitar obstáculos estáticos no previstos en el mapa, habiendo probado el desempeño de la red neuronal.

Capítulo 2

Modelos de Inteligencia Artificial aplicados a robots móviles

Un robot obtiene datos a través de sus sensores, los interpreta, toma decisiones y envía órdenes a sus actuadores. La "inteligencia" del robot depende de los algoritmos que le permiten interpretar los datos y tomar decisiones, por ésto el controlador del robot debe basarse en técnicas de Inteligencia Artificial (IA). Desde que nació la IA como disciplina ha existido la intención de realizar robots autónomos verdaderamente inteligentes, ésto es, robots que realizen tareas de manera similar a como las realizaría un ser humano.

En este capítulo se tratan algunas técnicas de inteligencia artificial que se pueden aplicar a la solución de algunos de los problemas de los robots móviles de manera eficaz.

2.1 La arquitectura de control

El controlador es el encargado de interpretar los datos que un robot recibe mediante sus sensores y de tomar decisiones para que realize los movimientos adecuados con el fin de cumplir con su tarea. La manera en que se distribuyen los componentes del controlador determina el comportamiento que tiene el robot y la velocidad a la que se pueden ejecutar las órdenes.

En los primeros días de la IA se propuso una arquitectura de controlador en la que se dividía la carga de trabajo en varios sistemas dispuestos uno tras otro: percepción, modelado del mundo, planificación y ejecución. A fines de los 80's se propuso una nueva arquitectura con varios módulos generadores de comportamiento, cada uno de los cuales tendría sus propios mecanismos de percepción modelado y planificación, y un árbitro que decide a qué comportamiento obedece el robot en un momento determinado de acuerdo a las circunstancias.

2.1.1 El enfoque tradicional

Los primeros robots "inteligentes" que fueron construidos a fines de los 60's y principios de los 70's, marcaron la pauta del desarrollo en robótica al menos por los siguientes 15 años. Shakey, por ejemplo, es un robot desarrollado en el Instituto de Investigaciones de Stanford (Stanford Research Institute) [2] contaba con un medio ambiente preparado especialmente para él, una cámara de televisión como sensor primario, una computadora externa para analizar las imágenes en base a descripciones hechas en la forma de cálculo de predicados de primer orden, que servían para generar una secuencia de acciones con un planificador que a su vez enviaba comandos a los actuadores que contaban con realimentación en base a otros sensores como el odómetro y una barra sensor de contacto.

En la mayoría de estos sistemas existía la tendencia a utilizar la visión por computadora solamente para reconstruir un mundo tridimensional a partir de imágenes bidimensionales. La inteligencia artificial era utilizada para realizar descripciones del mundo y manipularlas, y casi nunca se traducía en manipulaciones reales de los objetos reales [2]. La robótica sólo tenía como misión la interacción física con el mundo, los problemas principales eran: la planificación de una ruta libre de colisiones; la cinemática y dinámica directa; y la cinemática y dinámica inversa.

Como se puede ver en la figura 2.1 la interacción entre los componentes en el enfoque tradicional era:

1. Obtener datos de los sensores.
2. Interpretar esos datos.
3. Realizar un modelo del mundo en función de la respuesta de los sensores.
4. Planificar los movimientos necesarios en función de la meta y del modelo del mundo.
5. Enviar instrucciones a los actuadores.

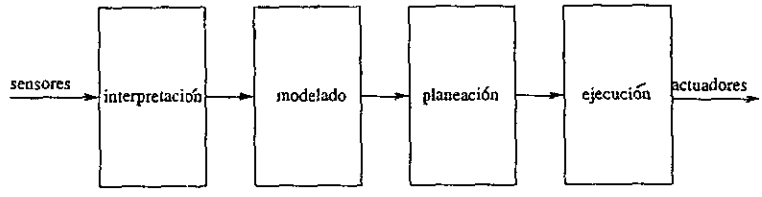


Figura 2.1: Enfoque tradicional

Este enfoque fue aplicado con relativo éxito en muchos casos, pero desgraciadamente en el momento en que los sistemas de prueba se han sacado de los ambientes fabricados expresamente para su operación, han fallado. Ahora es

claro que es muy difícil obtener suficientes datos de los sensores que permitan realizar un modelo exacto del mundo. Además el ciclo sensado, modelado, planificación, actuación es muy lento y puede hacer que un robot tarde varios minutos en tomar una decisión simple si no cuenta con suficiente potencia de cómputo (y en ocasiones aún teniéndola).

2.1.2 El enfoque de comportamientos

Alrededor de 1984, de manera independiente, varias personas comenzaron a reestructurar la organización de la inteligencia en los robots basados en los siguientes argumentos:

1. La mayor parte de las cosas que realiza la gente en su vida cotidiana no es resolver problemas o planificar, sino que realizan actividades rutinarias en un mundo dinámico.

2. La representación que un agente tiene del mundo, no necesariamente debe basarse en nombrar esos objetos con símbolos que posee el agente, sino que puede estar definido en términos de las interacciones del agente con el mundo.

3. Un observador puede hablar legítimamente de los objetivos y las metas de un agente, e incluso un agente no necesita manipular estructuras de datos simbólicas en el momento de actuar. Dado esto se puede compilar una especificación simbólica del agente antes de ser usado, logrando programas eficientes.

4. Es imposible obtener modelos internos del mundo que sean representaciones completas de ambientes externos y esto es totalmente necesario para que un agente actúe de manera competente.

5. Para poder evaluar la inteligencia de un robot, es necesario construir agentes completos que operen en un mundo dinámico usando sensores y en tiempo real.

6. La inteligencia puede emerger de componentes independientes que interactúan con el mundo.

Uno de los resultados más importantes de estas apreciaciones fue la arquitectura de subsumisión (subsumption architecture) que propuso Rodney Brooks [2]. Esta arquitectura está compuesta por una red de máquinas de estado finito aumentadas (AFSM's por sus siglas en inglés) generadoras de comportamientos, cada una de las AFSM's envía y recibe mensajes y almacena información respecto a los mensajes que se recibieron más recientemente. Dependiendo de los datos que cada AFSM recibe, puede enviar mensajes de salida o mensajes a otros AFSM's. Aunque la red es asíncrona, puede existir un AFSM que tenga un reloj interno para poder trabajar en el tiempo en que el mundo real "cambia". Los mensajes de salida generados por las diversas AFSM's son recibidos

por otro módulo diseñado para manejar los conflictos que se pueden producir. Por ejemplo un comportamiento puede ordenar un giro a la derecha, mientras la orden de otro comportamiento puede ser girar hacia la izquierda. La manera en que el conflicto se resuelve es dando prioridad a las órdenes dadas por el comportamiento más importante "sometiendo" a las demás momentaneamente. En la figura 2.2 se puede ver la distribución de los módulos generadores de comportamientos.

En el enfoque de comportamientos, se aprovecha la capacidad de generación de comportamientos de cada módulo, de manera que cuando se cuenta con varios de éstos, surgen comportamientos complejos a partir de otros que son muy simples, de esta forma se puede generar un comportamiento aparentemente inteligente.

A partir de que fue propuesto este modelo se han realizado muchos sistemas que demuestran que es viable construir inteligencia de esta manera [1]. Por otro lado, se ha considerado como una alternativa utilizar una mezcla de los dos enfoques aquí descritos, en éste se contaría con un sistema de alto nivel para planificar secuencias de actividades de bajo nivel que a su vez son realizadas por una arquitectura de comportamientos.

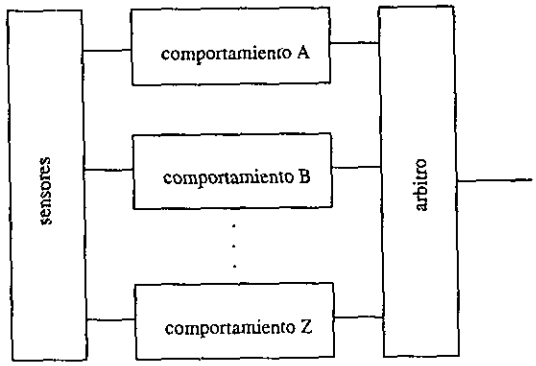


Figura 2.2: Arquitectura de comportamientos

2.2 Búsqueda de soluciones para encontrar un camino al objetivo

Una de la tareas centrales en un robot móvil es situarse en una configuración predeterminada, para lograrlo es necesario planificar la secuencia de configuraciones que debe adquirir con el fin de llegar al objetivo. Si se cuenta con un modelo adecuado del entorno en donde trabaja el robot, esto se puede hacer antes de iniciar cualquier movimiento utilizando un planificador. El principal componente del planificador es un mecanismo de búsqueda para encontrar un camino óptimo que el robot pueda seguir sin chocar.

El camino óptimo es aquel que permite minimizar o maximizar algún factor que se considere importante, por ejemplo el tiempo, la energía o la distancia. No siempre es necesario encontrar el mejor camino, ya que puede ser suficiente encontrar un camino satisfactorio o incluso puede bastar con encontrar al menos uno, sobre todo cuando el espacio de búsqueda es grande.

El problema básico del planificador consiste en que dado un punto inicial y un punto final, así como un mapa de nodos y conexiones que representan el entorno del robot, se debe encontrar algún camino o el mejor camino (quizá el más corto) para llegar del punto inicial al punto final.

Realizar un mapa, como se verá más adelante, es un problema fundamental de la planificación de rutas. En este mapa todos los nodos son puntos en los cuales puede estar el robot sin estar superpuesto con algún objeto de su área de trabajo. Además, tanto el punto inicial como el punto final son considerados nodos del mapa (si el punto final cumple con la condición de no superponerse con algún objeto), o de lo contrario la búsqueda no se puede realizar. Sólo existe conexión entre dos nodos si no hay ningún objeto que cruze la linea recta que los une, y el peso asociado a la conexión es igual a la distancia entre ellos. Debido a esta característica, a éste mapa se le conoce como mapa de caminos o de carreteras, y de manera más formal se puede decir que es un grafo de visibilidad. En la figura 2.3 se observa un ejemplo de grafo de visibilidad.

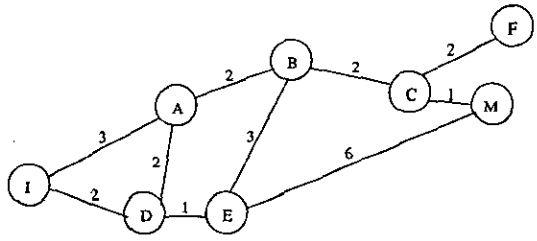


Figura 2.3: Mapa de caminos

El camino encontrado en la búsqueda es una secuencia de nodos a los cuales se puede llegar sin riesgo de tener una colisión con los objetos que se consideraron para determinar los nodos y las conexiones entre ellos. Además se pretende que éste camino sea el camino más corto o al menos uno de los más cortos.

Una forma eficiente de resolver este problema es utilizando un árbol de búsqueda, que es una colección de nodos (que podrían repetirse) y ramas. Los nodos representan trayectorias y las ramas conectan trayectorias a extensiones de trayectorias logradas en un solo paso. El árbol también tiene mecanismos para producir una descripción de trayectoria y para conectar la trayectoria con su correspondiente descripción [3].

Los árboles de búsqueda se usan en problemas de naturaleza diversa, para generar rutas de robots se puede utilizar un árbol para representar al mapa de caminos y proceder a una búsqueda en el mismo, como en la figura 2.4 en la que se vé el árbol generado a partir del grafo de la figura 2.3.

Existen diversos métodos para realizar búsqueda en un árbol. Se puede hacer una clasificación de éstos en dos tipos generales: los métodos ciegos y los métodos heurísticos. Los primeros realizan una búsqueda sin tener ninguna clase de información acerca del problema y se conforman con llegar al nodo meta, en tanto que los métodos heurísticos aprovechan las características del problema para hacer eficiente la solución y la búsqueda.

Además de que se debe tratar de encontrar una solución eficiente al problema que se plantea, también se debe cuidar la eficiencia de la búsqueda, ya que si consideramos que cada nodo del árbol tiene en promedio b ramas y que la profundidad del árbol es d, entonces el número total de trayectorias en él será b^d. El número de trayectorias se incrementa exponencialmente a medida que la profundidad del árbol aumenta, y por lo tanto la potencia de cómputo necesaria para resolver el problema también crece.

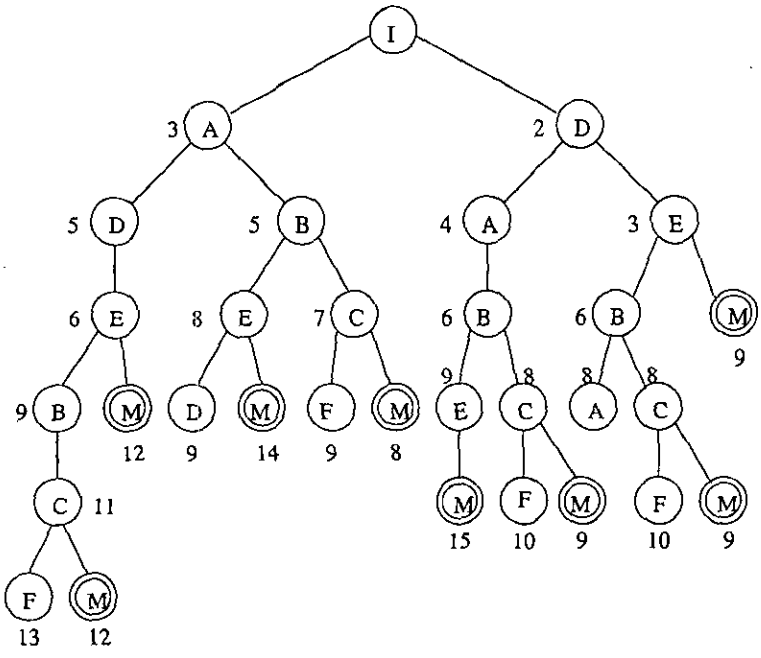


Figura 2.4: Representación del mapa de caminos de la figura 2.3 en un árbol

2.2.1 Métodos ciegos

Existen dos métodos ciegos: el de búsqueda en profundidad y el de búsqueda en amplitud. En ambos se realiza una búsqueda a prueba y error, por lo que el proceso es muy ineficiente, además que de no haber solución, es necesario recorrer todo el árbol para darse cuenta.

El método de búsqueda en *profundidad* consiste en partir del nodo origen y crear todas las trayectorias que se puedan generar a partir de él, revisar si una de

éstas trayectorias lleva al nodo destino, si nó tomar a uno de los hijos generados y repetir la búsqueda de la trayectoria a partir de él, las demás alternativas del mismo nivel se ignoran a menos que se hayan agotado las posibilidades de llegar al nodo destino a partir de la primera elección. Se puede fijar una profundidad límite cuando se desconoce la profundidad máxima del árbol En la figura 2.5 se puede observar un ejemplo de búsqueda en profundidad.

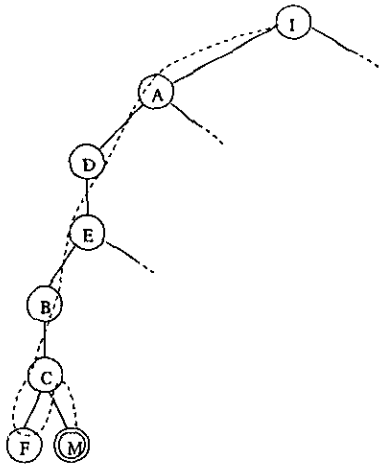


Figura 2.5: Búsqueda en profundidad

El método de búsqueda en amplitud, como se puede ver en la figura 2.6 consiste en obtener todos los hijos de todos los nodos del mismo nivel, si ninguno de ellos es el nodo destino, se incrementa el nivel y se repite el proceso hasta llegar a la solución o recorrer todo el árbol.

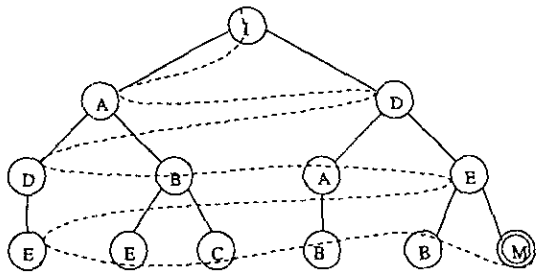


Figura 2.6: Búsqueda en amplitud

2.2.2 Métodos heurísticos

El objetivo primordial de los métodos heurísticos es aumentar la calidad de la búsqueda, utilizando una regla heurística (o de "sentido común") para reducir el

número de nodos a través de los cuales ésta se tiene que realizar. Dependiendo de la naturaleza del problema la regla heurística puede variar, en el caso de planificación de rutas de robots se puede considerar la distancia en línea recta del nodo que se evalúa hasta el nodo destino. En la figura 2.7 se observa la distancia en línea recta desde todos los nodos hasta el nodo meta *M*.

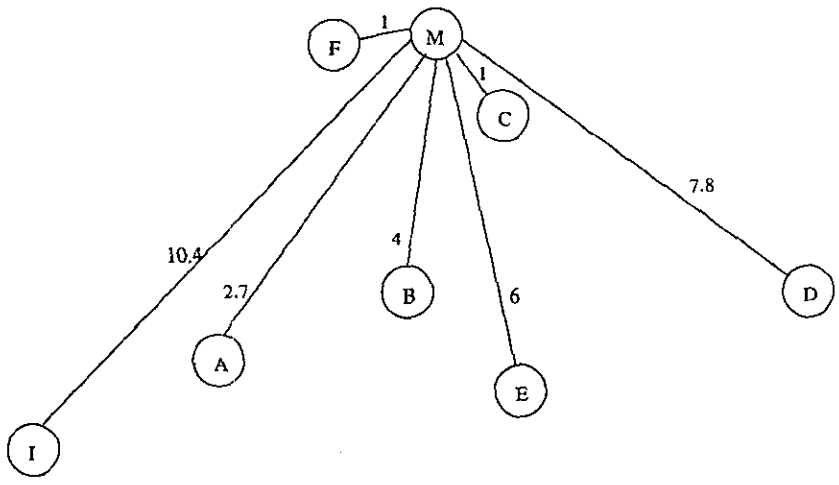


Figura 2.7: Distancias desde el nodo meta a todos los demás

Ya que la regla heurística mide qué tan bueno es un camino, estos métodos dan como resultado un buen camino o incluso el mejor. En primer lugar se muestran los métodos de *ascenso de colina*, de *búsqueda en haz* y de *búsqueda primero el mejor*, los cuales son mas eficientes que los métodos ciegos y localizan un buen camino, probablemente el mejor. En segundo lugar se describe el *método de la fuerza bruta*, el de *ramificación y cota*, de *ramificación y cota con estimación del límite inferior*, de *ramificación y cota con programación dinámica* y el *método A**, que garantizan encontrar la mejor solución con alta eficiencia.

El método de *ascenso de colina* (figura 2.8) es similar al de búsqueda en profundidad, con la diferencia que para decidir cuál es el nodo que se debe expandir, se extrae de los nodos una medida heurística de la distancia que queda por recorrer hasta la meta y se expande siempre el que proporciona la menor distancia.

El método de *búsqueda en haz o búsqueda rayo*, es similar a la *búsqueda en amplitud*, pero en lugar de expandir todos los nodos de un nivel sólo expande los w mejores. Si bien ésto puede reducir el número de nodos no ofrece aún un criterio adecuado que permita suponer que la búsqueda es eficiente. En la figura 2.9 se puede ver un ejemplo con $w = 3$.

La *búsqueda primero el mejor* sólo expande el nodo que ofrece la menor distancia sin importar en que nivel se encuentre, ésto mejora enormemente la eficiencia de la búsqueda, e incluso es probable que la ruta hayada sea la mejor, como en el ejemplo de la figura 2.10.

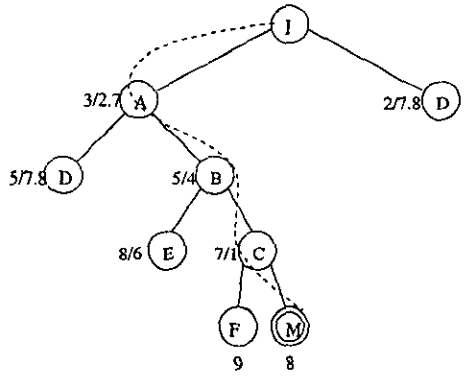


Figura 2.8: Búsqueda en ascenso de colina

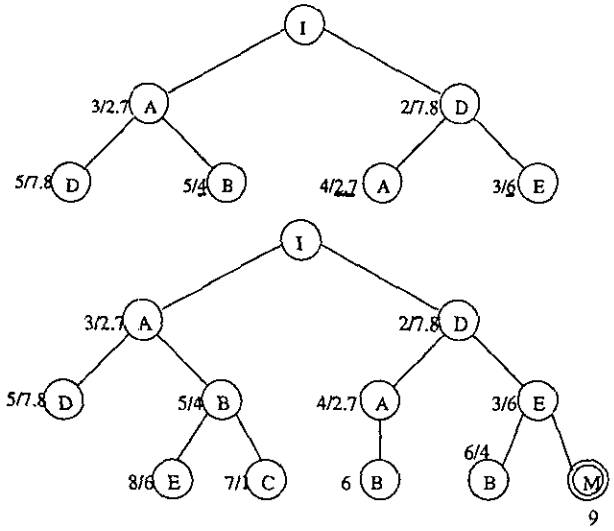


Figura 2.9: Búsqueda en haz con $w = 3$

El *método de la fuerza bruta* no es precisamente un método heurístico, ya que consiste en usar búsqueda en amplitud o en profundidad para encontrar todas las posibles soluciones al problema, y seleccionar la mejor de entre todas ellas. El costo de velocidad es enorme, sobre todo en problemas que tienen una gran cantidad de nodos y alta conectividad entre ellos.

La búsqueda de *ramificación y cota* realiza una búsqueda similar a *primero el mejor*, pero no se detiene en cuanto encuentra la primera solución. A partir de encontrar la primera solución continúa expandiendo nodos con el mismo método pero jamás expande un nodo cuya distancia sea mayor o igual a la menor distancia de las soluciones ya encontradas acotando la búsqueda. En la figura 2.11 se observan los primeros pasos de la búsqueda (incisos a al g) y el árbol después de haber finalizado (inciso h).

En la búsqueda de *ramificación y cota con estimación del límite inferior* se usa una fórmula heurística que hace una estimación de la distancia total a recorrer de la siguiente manera:

$$e(longitud\,total) = d(recorrida) + e(distancia\,restante)$$

Aquí el problema sería que la estimación de la distancia restante fuera mayor a la distancia restante real, por lo que se corre el riesgo de prohibir la expansión a nodos que lleven a la distancia óptima. Pero si en vez de hacer una estimación se hace una subestimación que garantize ser menor o igual a la distancia restante, no se corre el riesgo mencionado y sí se aumenta la eficiencia de la búsqueda. En el caso de robots una subestimación confiable es la distancia en línea recta desde el nodo en cuestión al nodo destino.

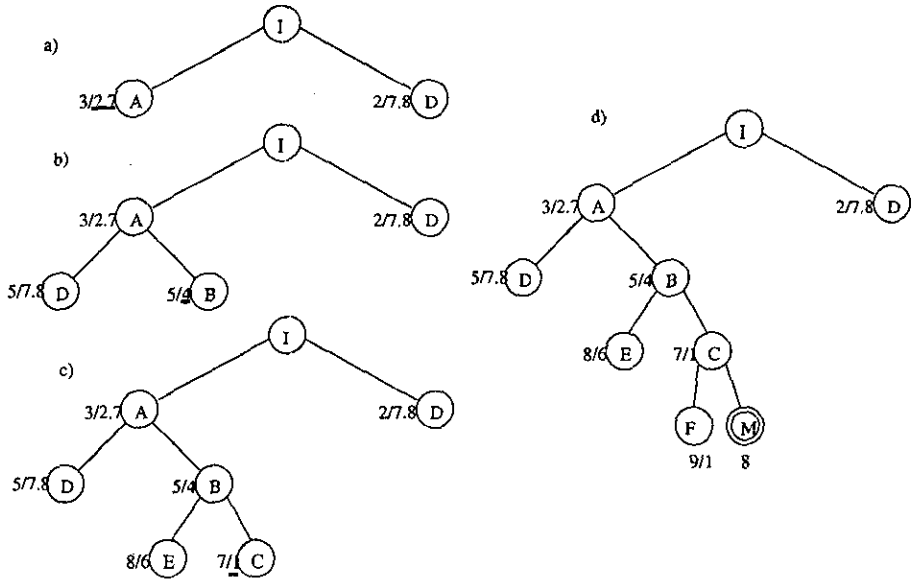


Figura 2.10: Búsqueda primero el mejor

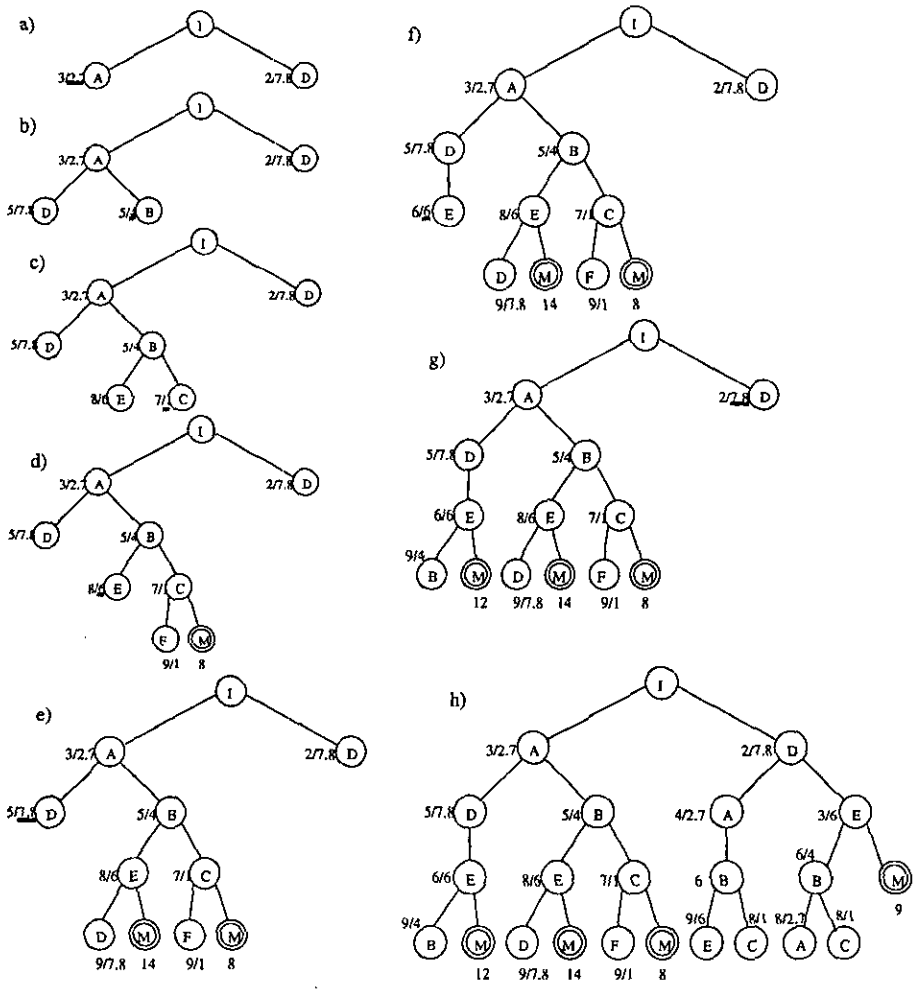


Figura 2.11: Búsqueda de ramificación y cota

El método de *ramificación y cota con programación dinámica* es similar al de *ramificación y cota*, pero parte del principio que dice que si existen dos formas de llegar al mismo lugar y una de ellas es menos costosa que la otra , no tiene sentido seguir evaluando opciones en la mas costosa. En éste método si un nodo se repite en alguna parte del árbol, no se debe seguir expandiendo aquel que implique un costo mayor. Ésto elimina búsquedas inútiles.

Finalmente, el *método A** es una combinación de los métodos de ramificación y cota con estimación de límite inferior y con programación dinámica. Este método es el más eficiente de todos los que aqui se han visto, como se puede ver en la figura 2.12 en la que en sólo cuatro pasos encuentra el mejor camino.

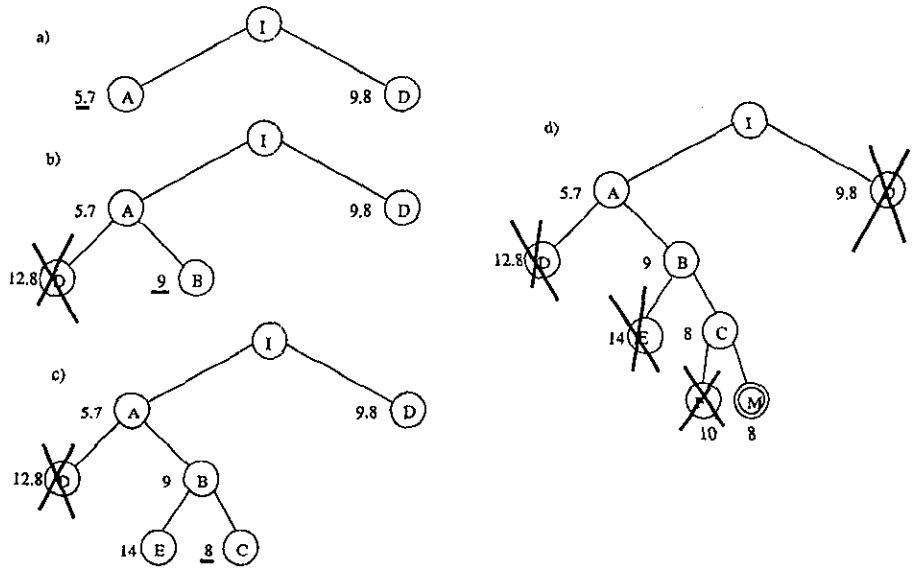


Figura 2.12: Busqueda A*

2.3 Sistemas Expertos

Un sistema experto (SE) es "un programa de computadora que utiliza conocimientos y procedimientos de inferencia para resolver problemas que son lo suficientemente difíciles para requerir capacidad experta humana para su solución" [4]. Se han propuesto varias maneras para representar el conocimiento y para implantar los procedimientos de inferencia sobre el mismo, pero el más popular es el modelo basado en reglas, que aquí se detalla. En este modelo el sistema trata de emular la capacidad de toma de decisiones de los seres humanos utilizando reglas, que forman el conocimiento y están almacenadas en una base de conocimientos. El funcionamiento de los SE's permite introducir al robot conocimientos de alto nivel para poder interactuar con su entorno.

2.3.1 Modelo y componentes

A partir de que Newell y Simon realizaron un programa al que llamaron "solucionador general de problemas", se pudo ver que el conocimiento necesario para resolver problemas se puede expresar en gran medida en forma de *reglas de producción*, del tipo si-entonces como en:

```
si
   antecedente 1    y
   antecedente 2    y

        ⋮

   antecedente n
entonces
   consecuente 1
   consecuente 2

        ⋮

   consecuente m
```

donde $n > 0$ y $m \geq 0$.

La idea esencial es que las entradas sensoriales que recibimos, provocan estímulos que activan las reglas apropiadas en nuestra memoria de largo plazo y a partir de ahí se produce la respuesta adecuada a la situación.

Un sistema experto trabaja con hechos que pueden considerarse como parte de una memoria de corto plazo y que se almacenan en la memoria de trabajo. Los antecedentes de las reglas son, generalmente, cuestionamientos acerca de la existencia o la inexistencia de hechos que coinciden con algún patrón predefinido. Los consecuentes son acciones, que pueden ser llamadas a funciones o instrucciones para aseverar o negar hechos en la memoria de corto plazo. En la figura 2.13 se pueden ver los componentes de un sistema experto.

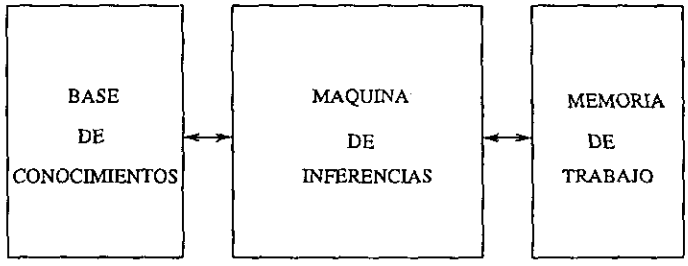


Figura 2.13: Esquema general de un sistema experto

Además de los hechos y las reglas, los sistemas expertos cuentan con un mecanismo para hacer inferencias, que es conocido como máquina de inferencias.

A través de ésta el sistema realiza inferencias tratando de emular el razonamiento que sólo es propio de los seres humanos. El proceso de inferencia consiste básicamente en encontrar las reglas válidas en cierto momento y ejecutar las acciones correspondientes a alguna de ellas.

La máquina de inferencias puede actuar de dos maneras: en la primera busca los hechos que hay en la memoria de trabajo y después verifica qué reglas de la base de conocimientos cumplen con ellos; en la segunda, busca los antecedentes de las reglas y los coteja con los hechos que hay en la memoria de trabajo. Todas las reglas que resultan del proceso anterior, se consideran reglas activas y susceptibles de ser ejecutadas, pero la máquina de inferencias tiene que elegir sólo una de ellas para ejecutarla. La elección se puede realizar tomando en cuenta cualquier criterio y al proceso de ejecución se le llama disparo.

Aunque las maneras de hacer un sistema experto pueden ser diversas, típicamente sus componentes, como se aprecia en la figura 2.14 son los siguientes [5]:

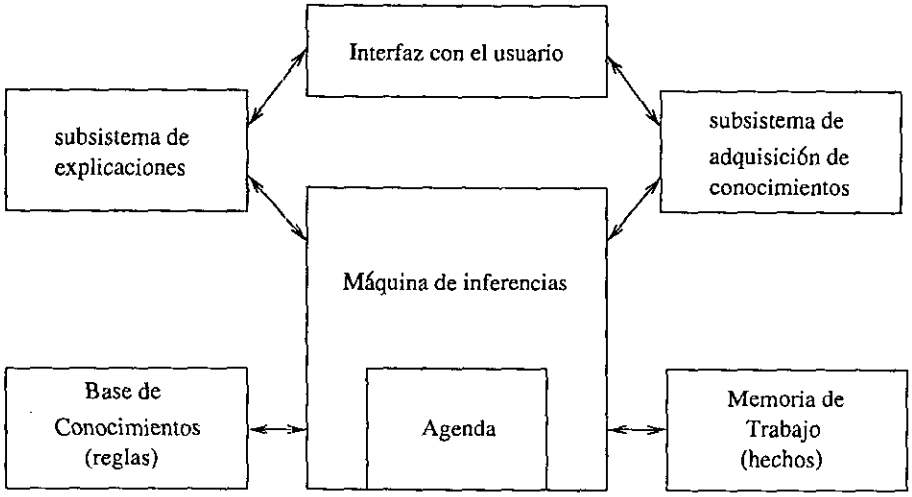


Figura 2.14: Estructura de un sistema experto basado en reglas

Interfaz con el usuario Mecanismo de comunicación entre el usuario y el sistema experto.

Subsistema de explicaciones Le explica al usuario el proceso de inferencias realizado para llegar a una conclusión.

Subsistema de adquisición de conocimientos Permite que el usuario introduzca conocimientos al sistema.

Base de conocimientos Base de datos global que almacena las reglas.

Memoria de trabajo Base de datos global que almacena los hechos usados para activar las reglas.

Máquina de inferencias Realiza inferencias para determinar qué reglas son satisfechas por hechos u objetos, de éstas ejecuta la que tiene mayor prioridad.

Agenda Es una lista priorizada de reglas creada en el proceso de inferencias, cuyos patrones son satisfechos por hechos u objetos en la memoria de trabajo y que por lo tanto son susceptibles de ejecutarse.

2.3.2 Entornos de desarrollo

Existen lenguajes de programación que permiten resolver problemas basados en la programación procedural y otros que facilitan el desarrollo de aplicaciones para inteligencia artificial, pero niguno de ellos proporciona maneras flexibles y robustas para representar el conocimiemto. Debido a ésto, se hace necesario contar con un lenguaje de programación orientado a los sistemas expertos que permita dos niveles de abstracción: la abstracción de datos y la abstracción de conocimientos.

Por otro lado, en los lenguajes procedurales existe una estrecha relación entre los datos y los métodos para manipularlos por lo cual, el programador debe ser muy cuidadoso al describir la secuencia de ejecución, en un lenguaje de sistemas expertos se requiere un control de ejecución mucho menos rígido debido a que los datos (hechos) y el conocimiento (reglas) están separados explícitamente. Para aplicar los conocimientos a los datos se utiliza una pieza de código totalmente distinta: el motor de inferencias. Esta separación permite un altísimo grado de paralelismo y modularidad.

A partir de los años 70's el crecimiento explosivo en el área de los sistemas expertos, condujo a la existencia de un sinúmero de herramientas enfocadas a su desarrollo. Desde lenguajes como PROLOG que solo permiten alimentar instrucciones simples, hasta herramientas que incluso le permiten al usuario introducir el conocimiento a través de ejemplos almacenados en tablas u hojas de cálculo y que generan el código por sí mismos. Por otro lado, también se han desarrollado *shells*, que son el producto de aplicaciones específicas a las cuales se les extrajo el conocimiento para poder darle una aplicación diferente.

Una herramienta muy útil para el desarrollo de sistemas expertos es CLIPS (C language Integrated Production System) [6]. Esta herramienta fué desarrollada en el Centro Espacial Johnson de la NASA a partir de 1984 para profundizar en el conocimiento de la construcción de herramientas para sistemas expertos. La primera versión que fué puesta a disposición de usuarios fuera de la NASA fué la 3.0 que apareció en el verano de 1986. Actualmente se usa la versión 6.0 que apareció en 1993.

CLIPS es una herramienta que cuenta con una base de conocimientos en la que se introducen los conocimientos en forma de reglas de producción. También tiene una memoria de trabajo en la que se almacenan conocimientos en forma

de hechos. La máquina de inferencias que utiliza utiliza el algoritmo de Rete que hace que la búsqueda de las reglas activas sea muy rápida.

CLIPS también permite introducir segmentos de código procedural y orientado a objetos. Cuenta con una interfaz de texto y otra de menues que permiten cargar reglas, hechos, funciones, objetos y monitorear el funcionamiento de los programas. La misma interfaz sirve como mecanismo de explicaciones, por lo que es fácil depurar reglas y evaluar el sistema bajo condiciones controladas.

CLIPS se distribuye a un costo muy bajo y con el código fuente en C incluido. Esto le permite al usuario agregar al sistema experto funciones en lenguaje C con sólo agregarle unas cuantas líneas y volverlo a compilar.

2.4 Redes Neuronales Artificiales

El principal objetivo de la Inteligencia Artificial es construir máquinas que emulen al ser humano en actividades que requieren inteligencia. Se sabe que el origen de la inteligencia humana es el cerebro y que está compuesto por millones de neuronas que están interconectadas formando una inmensa red, de ahí que resulte natural la idea de reproducirlas para producir inteligencia.

Las redes neuronales artificiales se inspiran en este modelo formando un sistema de cómputo compuesto por un grán número de elementos de proceso simples (neuronas) que están altamente interconectados.

2.4.1 Definición

Según Simon Haykin [7], una red neuronal artificial es un procesador masivamente paralelo que almacena conocimiento adquirido mediante experiencia y lo hace disponible para ser utilizado, la red adquiere el conocimiento a través de un proceso de entrenamiento y los conocimientos se almacenan en forma de ponderaciones, también conocidas como pesos sinápticos, aplicadas a las conexiones entre las neuronas que la conforman.

Usualmente, las redes neuronales se implementan mediante componentes electrónicos y se simulan mediante software en computadoras digitales. Aunque si la red neuronal se simula con software, se pierde su paralelismo intrínseco.

Las principales características de las redes neuronales artificiales son:

No linealidad. Cada neurona en la red, como se verá más adelante, tiene una respuesta no lineal, consecuentemente la red neuronal es no lineal.

Mapeo Entrada-Salida. Durante el entrenamiento de las redes neuronales que aprenden mediante *aprendizaje supervisado*, se les muestran ejemplos de entradas y la salida deseada correspondiente, y se modifican los pesos sinápticos de manera que la diferencia entre las salidas de la red y las entradas asociadas se minimize, de acuerdo a un criterio estadístico. De esta forma la red aprende de los ejemplos mediante la construcción de un mapeo Entrada-Salida. Este mapeo permite hacer una generalización en el sentido de que cuando a la red neuronal ya entrenada se le muestran

entradas que no se le mostraron durante su entrenamiento, ésta genera una salida correspondiente al mapeo construido.

Adaptividad. Durante el entrenamiento, la red neuronal *adapta* sus pesos sinápticos para poder entregar las respuestas deseadas, de esta manera no es necesario encontrar la solución algorítmica del problema. Incluso, si una red neuronal es entrenada para un conjunto de condiciones determinadas, puede ser reentrenada para adaptarse a nuevas condiciones.

Tolerancia a fallas. Cuando una red neuronal está construida con circuitos electrónicos, presenta una gran tolerancia a fallas, en el sentido de que su desempeño se degrada suavemente ante condiciones adversas.

Operación en tiempo real. La naturaleza paralela de las redes neuronales hace que presenten resultados con un retardo de tiempo muy pequeño, por lo que se pueden utilizar en aplicaciones que requieren resultados en tiempo real.

2.4.2 Modelo de neurona

Una neurona, como se observa en la figura 2.15 es una unidad de procesamiento compuesta por los siguientes elementos:

- Un conjunto de p sinapsis o conexiones, caracterizadas por un peso. Una señal x_j a la entrada de la sinapsis j conectada a la neurona k se multiplica por el peso w_{kj}. En w_{kj} el primer subíndice corresponde a la neurona en cuestión, y el segundo corresponde a la sinapsis a la que corresponde el peso. Si w_{kj} es positivo, la sinapsis asociada es de excitación, si es negativo, la sinapsis es de inhibición, en caso de que valga cero, es equivalente a la ausencia de sinapsis.

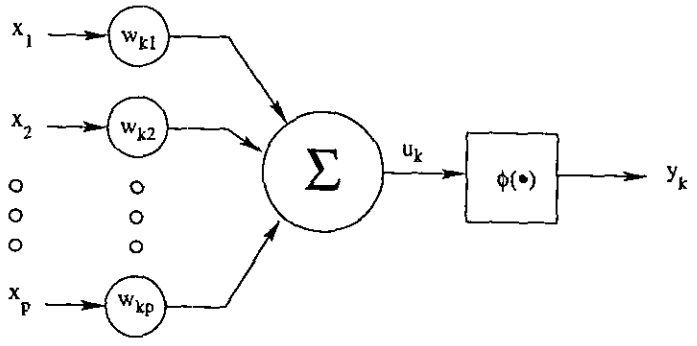


Figura 2.15: Neurona U_j

- Un sumador que suma todas las entradas de la neurona, una vez multiplicadas por sus correspondientes pesos. Esta operación es, de hecho una combinación lineal. El sumador entrega a su salida

$$u_k = \sum_{j=1}^{p} w_{kj} x_j$$

- Una función de activación para limitar la amplitud de la salida de la neurona, manteniéndola en un rango predefinido, que típicamente está en el intervalo [0,1] o [-1,1]. La salida de la neurona es $\phi(u_k)$

2.4.3 Función de activación de la neurona

La función de activación en la mayoría de los casos es cualquiera de las siguientes:

Función escalón cuya salida es 1 si la suma de las entradas es mayor o igual al umbral de la neurona, y 0 o -1 si la suma es menor al umbral.

Función lineal y mixta en la que la salida es igual a una función lineal de la suma de las entradas si ésta se encuentra entre dos límites predefinidos, es 0 o -1 si la suma es menor que el límite inferior y es 1 si la suma es mayor que el límite superior.

Función continua que puede ser cualquier función definida para todo el intervalo de posibles valores de entrada, con incremento monotónico y límites superior e inferiores. La función contínua más usada es la sigmoide

$$f(x) = \frac{1}{1 + e^{-\alpha x}}$$

donde α es el parámetro de pendiente de la sigmoide. Esta función tiene la ventaja de que es diferenciable y que su derivada es siempre positiva y cercana a cero en valores grandes, ya sean positivos o negativos, lo cual es muy importante durante el proceso de entrenamiento.

Función gaussiana que es una campana gaussiana típica con centro y ancho adaptables.

2.4.4 Arquitectura de la red neuronal

La estructura de la red neuronal, esto es, la forma en que se conectan las neuronas, está ligada con el algoritmo que se utiliza para su entrenamiento. A esta estructura se le conoce como *arquitectura*. Existen varias arquitecturas para redes neuronales, las principales son:

Redes de una capa con alimentación hacia adelante. Una red neuronal puede organizarse en capas. Cuando sólo hay una capa, se tiene una capa

de entradas que son alimentadas a la capa de salida, donde se encuentran los nodos de cómputo que calculan las salidas de la red. Como la alimentación es hacia adelante, no existe retroalimentación.

Redes multicapa con alimentación hacia adelante. Son similares a las redes de una capa, pero cuentan con una o más capas *ocultas* en las que también hay elementos de procesamiento. Las salidas de las neuronas en una capa son entradas de la siguiente capa como se puede ver en el ejemplo de la figura 2.16.

Redes Recurrentes. Estas redes se diferencían de las de alimentación hacia adelante en que al menos existe una conexión de realimentación.

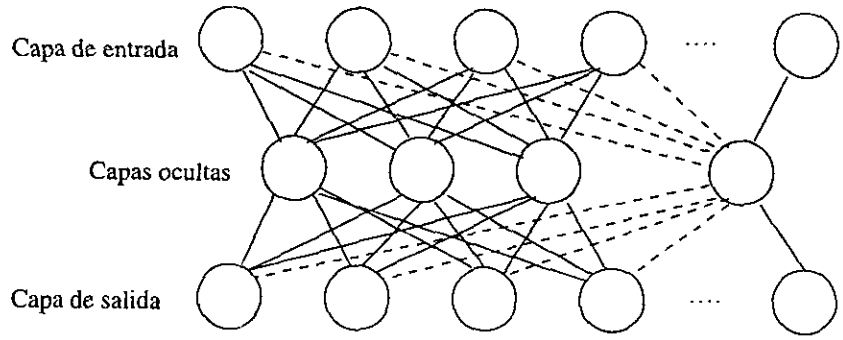


Figura 2.16: Red Neuronal

Existen varias aplicaciones de redes neuronales en robots móviles, para identificar objetos en escenas de video, para calcular distancias a objetos, etc.

2.5 Agentes

El concepto de Agente Inteligente Autónomo [8] es muy reciente y da respuesta a un conjunto de problemas planteados por la Inteligencia Artificial Distribuida. Se puede decir que un agente es un dispositivo, o un programa que cuenta con las siguientes características:

Autonomía No requieren de la intervención directa de una persona o programa externo, controlan sus propias acciones y su estado interno.

Capacidad social Interactúan con otros agentes a través un protocolo de comunicación.

Reactividad Perciben su entorno y responden oportunamente a los cambios que éste presente.

Pro-actividad Tienen un comportamiento dirigido a metas en el que toman la iniciativa para cumplir con ellas.

Existen otras características que algunos autores asocian al concepto de agente, dependiendo del contexto en que sean utilizados. Por ejemplo, la Inteligencia Artificial hace énfasis en conceptos mentales, como el conocimiento o las creencias. Las principales ventajas de los agentes son que pueden planificar y ejecutar acciones en base a su estado interno, a sus metas, a sus hipótesis y a sus conocimientos sobre otros agentes, pueden colaborar con otros agentes para tomar decisiones en consenso y en ocasiones permiten crear otros agentes.

El desarrollo de sistemas inteligentes que realizan un modelo del mundo en base a la información que obtienen de sus sensores, hacen planes y los ejecutan, es insuficiente cuando se enfrenta al enorme conjunto de incertidumbres que presenta el mundo real. Es por ésto que se han propuesto soluciones distribuidas que trabajen en forma cooperativa, principalmente en aplicaciones que están separadas geográficamente.

Para apoyar la toma de decisiones en sistemas distribuidos, se necesita un mecanismo de comunicación que permita que todos los componentes cumplan con su trabajo de manera eficiente. Hay dos clases de estructuras de comunicación:

Sistemas de pizarrón La comunicación se facilita aprovechando una estructura de conocimiento llamada pizarrón, en la que todos pueden colocar y leer información.

Sistemas de transferencia de mensajes Cuando un módulo requiere que otro realize o se entere de algo, le manda un mensaje. Todos los módulos tienen previo conocimiento de los nombres de los demás para poder mandar los mensajes al destinatario.

Los sistemas de pizarrón están compuestos por módulos independientes que generan conocimientos, un pizarrón que les sirve para comunicarse y un sistema de control del pizarrón, que determina la secuencia en que los generadores de conocimientos operan en el pizarrón. Este método apoya la estructura modular, pero al centralizar la comunicación con el pizarrón, propicia que el sistema se embotelle.

Los sistemas de transferencia de mensajes pueden basarse en actores o en agentes. Un actor es un objeto activo que procesa mensajes, crea nuevos actores o los activa. Las acciones del actor están definidas por un repertorio que describe su comportamiento, este puede ser: modificación de estado, actualización de lista de destinatarios, envío de mensajes o creación de nuevos actores. La comunicación entre los actores es asíncrona, por lo cual una vez que es enviado un mensaje, se pasa a procesar el siguiente sin esperar respuesta.

Los diferentes componentes de un robot móvil se pueden implantar como módulos que se comunican mediante una estructura de pizarrón y el robot se puede ver como un agente que podría comunicarse con otros robots (agentes también) o personas o programas en una red (que por supuesto serían también agentes autónomos).

Capítulo 3

Planificación de movimientos y navegación

Un robot es un dispositivo que opera en el mundo físico el cual está poblado por objetos que limitan sus posibilidades de movimiento. La secuencia de acciones necesaria para concluir una tarea está determinada por la naturaleza de la misma y por los objetos que "estorban" para realizarla. Este problema es enfrentado con técnicas de planificación movimientos y de navegación. Una organización lógica típica para un robot móvil se muestra en la figura 3.1. La misión de cada módulo se lista a continuación.

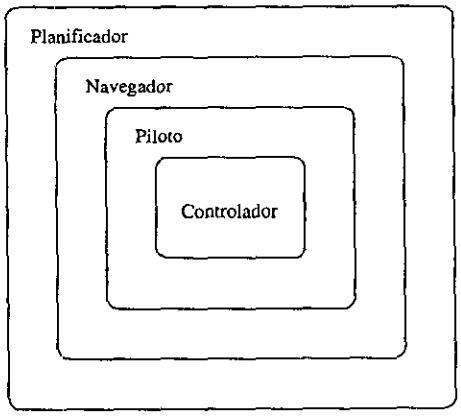


Figura 3.1: Organización lógica de un robot móvil

Planificador Determina una ruta óptima global, de acuerdo a algún criterio de optimización, para que el robot alcanze su configuración meta, utilizando conocimiento *a priori* del entorno del robot y de las restricciones a las que

está sujeto. También puede definir un tiempo máximo para concluir la misión.

Navegador Divide la ruta seleccionada por el planificador en segmentos independientes y encuentra un movimiento contínuo para seguirlas, proporciona al piloto las distancias, velocidades y ángulos de giro que tendrá que realizar.

Piloto Sigue la ruta definida por el navegador, ligando la variable tiempo a la misma y evita obstáculos detectados con los sensores del robot. La combinación de una ruta y "marcas de tiempo sobre la misma da como resultado una trayectoria. Para generar la trayectoria se debe considerar la dinámica del robot.

Controlador Obedece al piloto para que los motores cumplan con los movimientos esperados y obtiene datos de los sensores.

Se puede ver que este esquema se ajusta al enfoque tradicional del que se habló en el capítulo anterior, pero se puede adaptar para operar en un esquema que aproveche las ventajas del enfoque tradicional realizando planificación de movimientos de alto nivel y utilizando un navegador y un piloto basados en comportamientos.

3.1 El problema básico de planificación

Para hacer planificación de movimientos se considera que el robot es el único objeto en movimiento y se ignoran sus propiedades dinámicas, con el fin de evitar consideraciones temporales. Se asume que los movimientos que se realizan están libres de contacto, evitando así modelar la interacción mecánica que se podría dar. En resumen, el problema de planificar movimientos físicos se transforma en un problema de planificación de rutas puramente geométrico en el que se cumple que:

- El robot A se mueve en un espacio Euclidiano W, llamado espacio de trabajo, que pertenece a $\Re^N$, donde $N = 2$ o 3.
- Los obstáculos $B_1, \ldots, B_q$ son objetos rígidos fijos en W.
- La geometría de $A, B_1, \ldots, B_q$ y la posición de los $B_i's$ en W se conocen con precisión.
- No existen restricciones cinemáticas que limiten los movimientos de A (el robot es un objeto libre).

Con base en estas consideraciones se puede definir el problema básico de planificación de movimientos de robots de la siguiente manera [9]: Dada un posición y orientación inicial y una posición y orientación meta del robot A en el espacio W, se debe generar una ruta τ que especifique una secuencia continua de

posiciones y orientaciones del robot A que eviten el contacto con los obstáculos $B_i's$, partiendo de la posición y orientación iniciales y terminando en la posición y orientación metas. Si no existe tal ruta se reporta la imposibilidad de resolver el problema.

3.2 Espacio de configuraciones

El espacio de configuraciones es un concepto introducido por Lozano Pérez como una herramienta de representación [10]. En ésta el robot es tratado como un punto en un espacio llamado el *espacio de configuraciones del robot.* Los elementos geométricos y conceptos físicos se transforman para representarse como construcciones geométricas. Ésto permite formular el problema de planificación de movimientos con precisión y transformarlo en el problema más sencillo de planificar los movimientos de un punto y no de un objeto dimensionado.

Tanto el robot A como los obstáculos $B_1, \ldots, B_q$ son subconjuntos cerrados del espacio W. Se definen F_A y F_W como planos cartesianos ligados a A y a W respectivamente, F_A es un plano cartesiano que se mueve junto con A, y aunque todos los puntos pertenecientes al robot están fijos con respecto a F_A, su posición en W depende de la posición de F_A con respecto a F_W. Por su parte los $B_i's$ se consideran rígidos y fijos en el espacio W, por lo que tienen una posición fija con respecto a F_W.

La configuración de un objeto es una especificación de la posición de todos sus puntos con respecto a un marco de referencia fijo. De aquí se deduce que una configuración q del robot A es la posición τ y la orientación θ de F_A (el plano ligado al robot) con respecto a F_W (el plano ligado al espacio W). El *espacio de configuraciones* de A es el espacio formado por todas las configuraciones que puede adquirir A y se le denota como C. Al subconjunto de W ocupado por el robot A en la configuración q se le denota $A(q)$, y al punto a de A en la configuración q se le denota $a(q)$ en el espacio W.

Hay varias maneras de representar una configuración. Por ejemplo, para la posición se puede utilizar el vector de N coordenadas que representa al origen de F_A en F_W y la orientación se puede describir como una matriz de NxN cuyas columnas son las proyecciones de los vectores unitarios de los ejes de F_A en FW. La descripción aquí ejemplificada es redundante y se puede minimizar, pero ésto no siempre es conveniente, ya que aunque el espacio de configuraciones C se parece a un espacio de $\Re^m$, formalmente no lo es, dado que se descompone en una unión de copias ligeramente diferentes de R^m, entre las cuales se va desplazando el robot, dependiendo del tipo de movimiento que realice.

Para establecer los movimientos que debe realizar un robot para trasladarse de una configuración determinada a otra, se necesita seguir una ruta. Una ruta para el robot A desde $q_{inicial}$ hasta q_{meta} es una función contínua

$$\tau : [0,1] \rightarrow C \qquad \text{donde } \tau(0) = q_{inicial} \text{ y } \tau(1) = q_{meta}$$

$q_{inicial}$ y q_{meta} son, respectivamente, las configuraciones inicial y meta del la ruta.

Para que se pueda mover al robot, es necesario que exista una ruta entre al menos dos configuraciones.

3.3 Representación de los obstáculos

Los obstáculos $B_i, i = 1, 2, \ldots, q$ que están en W, se representan como regiones cerradas, aunque no necesariamente limitadas, de $\Re^N$. Estos $B_i's$ denotan tanto a los objetos físicos como a las áreas que ocupan. A la transformación de los obstáculos B_i en el espacio de configuraciones C se le llama *obstáculo-C_i*, que se denota como CB_i y se define como la región en C que cumple con:

$$CB_i = \{q \in C \mid A(q) \cap B_i \neq 0\}$$

Por ejemplo, si se tiene un robot cilíndrico que se mueve en un espacio de 2 dimensiones, y un obstáculo poligonal B_i, el *obstáculo-C_i* se obtiene haciendo que el área del obstáculo crezca isotrópicamente por el radio de A, como se ejemplifica en la figura 3.2.

Esta representación permite establecer rutas solamente en las partes del espacio W cuya intersección con los *obstáculos-C* es vacía. De la definición de los $CB_i's$ se observa que para construirlos se debe considerar la forma tanto del robot A como de los obstáculos B_i.

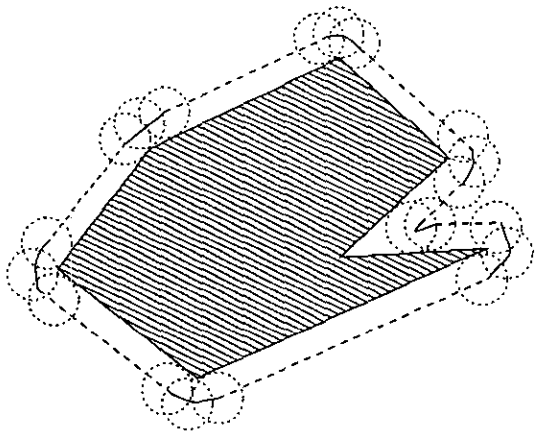


Figura 3.2: Expansión de un obstáculo polígonal para un robot cilíndrico

Hay varias maneras de formar el espacio de configuraciones, pero se puede hablar de dos casos generales: Cuando el espacio de trabajo del robot es de dos dimensiones y cuando es tridimensional.

3.3.1 Espacio de trabajo de dos dimensiones

Cuando el robot y los obstáculos son poligonales, los desplazamientos se realizan en un espacio de dos dimensiones y el robot sólo se puede trasladar, las $CB_i's$ se

forman a partir del área de contacto que se forma entre la superficie del robot y la de los polígonos, si se le pone a éste a desplazarse libremente sobre toda la superficie de trabajo sin que se presente alguno de los contactos que se listan a continuación:

Contacto tipo A que se dá cuando los interiores de A y B no se sobreponen.

Contacto tipo B que se dá cuando un vértice del robot está en el borde de de B sin sobreponerse a su interior.

Si se le permite girar al robot, se forma un espacio de tres dimensiones formado por planos. Cada plano se forma fijando la orientación del robot y procediendo de la forma descrita, por lo que entre plano y plano solo hay un diferencial de orientación.

3.3.2 Espacio de trabajo de tres dimensiones

Por otra parte cuando el robot se desplaza en un espacio de tres dimensiones, los obstáculos se representan mediante polihedros. Las regiones $CB_i's$ son de seis dimensiones que están limitadas por superficies de cinco dimensiones generadas por el contacto que se dá entre la superficie del robot. En este caso los contactos pueden ser:

Contacto tipo A que se dá entre una cara del robot y un vértice de B.

Contacto tipo B entre un vértice de A y una cara de B.

Contacto tipo C entre un borde de A y un borde de B.

El mecanismo que se sigue para formar las $CB_i's$ es similar al que se sigue en los espacios de dos dimensiones.

Los casos anteriores son casos particulares que se pueden generalizar si se representan los obstáculos y los objetos como subconjuntos semialgebraicos de $W = \Re^N$. Un conjunto semialgebraico se define mediante sentencias de lógica de primer orden cuyos predicados son $=, \neq, >, <, \geq$ y $\leq$, las variables son números reales y los términos son polinomios multivariables con coeficientes reales [9].

Si el robot y el conjunto finito de obstáculos se representan como subconjuntos semialgebraicos de $\Re^N$, todos los *obstáculos-C*, las áreas libres, las áreas de contacto y las áreas válidas son subconjuntos semialgebraicos de $\Re^{N(N+1)}$. Esta representación es en si misma una herramienta adecuada para operar con los objetos y determinar los caminos viables para un robot de manera eficiente. En la figura 3.3 se observan los elementos del espacio de configuraciones creado para un robot cilíndrico en un entorno poblado por polígonos y que solo puede tener desplazamientos y rotaciones en un plano. Ahí se pueden ver las fronteras de los *obstáculos-C* que ya fueron generados, por lo que el robot A ya se puede considerar solamente un punto que se mueve en el espacio C que es el área que está dentro del rectángulo y fuera de los *obstáculos-C*.

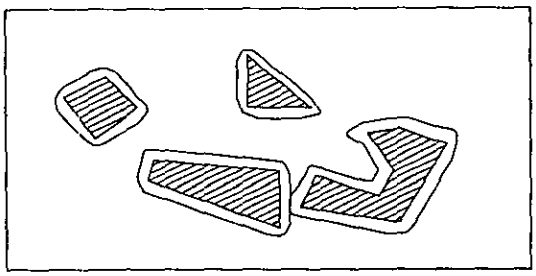


Figura 3.3: Espacio de configuraciones para un robot cilíndrico y obstáculos poligonales

3.4 Métodos de solución del problema básico de planificación

Una vez que se cuenta con una representación confiable del entorno en el que se desempeña el robot, se puede pasar al problema de planificación en si. Como ya se dijo se trata de encontrar una secuencia de rutas libres de colisión para alcanzar la meta. El planificador recibe las configuraciones inicial y meta y el tiempo para ejecutar el traslado, se conocen las restricciones del robot, aceleración y velocidad, se determinan las restricciones de posición y se planifica la ruta respetando algún principio de optimización.

Existen varias técnicas para resolver este problema, a continuación se describen los enfoques mas estudiados que son el de mapas de caminos, el de descomposición de celdas y el de campos potenciales.

3.4.1 Mapas de caminos

La idea subyacente en el método de *mapa de caminos* es capturar la conectividad del espacio libre para el robot en forma de una red de curvas de una dimensión, a esta red se le llama también R. Una vez construido, se utiliza como un conjunto de rutas estandarizadas. La planificación de rutas se reduce a conectar las configuraciones inicial y meta al mapa, y buscar en él un camino.

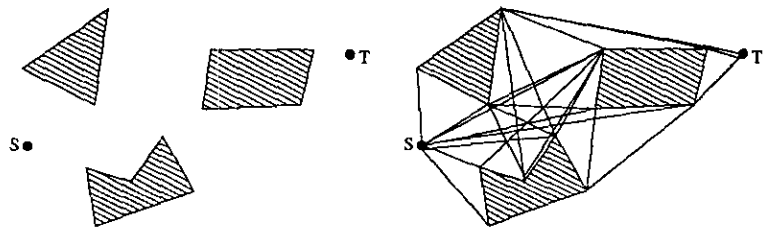


Figura 3.4: Grafo de visibilidad

Una forma de construir el mapa de caminos es mediante el *grafo de visibilidad* que se construye conectando todos los vértices de los límites del área libre mediante un segmento de recta, siempre que ésta no pase por el interior de algún *obstáculo C*. En la figura 3.4 se observa un ejemplo del uso del grafo de visibilidad.

Otra forma es definir una función continua de el espacio libre para formar R. Cuando el espacio es de dos dimensiones y con *obstáculos-C* poligonales, la función resultante se puede representar, por ejemplo, con el diagrama de Voronoi del espacio libre.

El diagrama de Voronoi (figura 3.5) es un subconjunto unidimensional del espacio libre que maximiza el espacio que queda entre el robot y los obstáculos. Usándolo se logra retraer el área libre hasta obtener solo una pequeña parte de la misma.

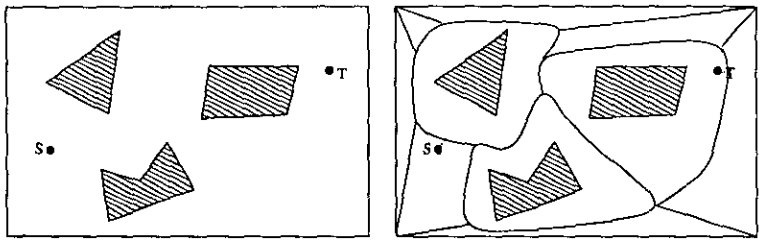


Figura 3.5: Diagrama de Voronoi

Otro método es el de vía rápida o *freeway*, que es similar al anterior. Se usa en robots poligonales que se trasladan y giran entre obstáculos también poligonales. Se restringe al punto de referencia del robot a estar sobre una red de segmentos de recta que es similar al diagrama de Voronoi.

Un método más, conocido como algoritmo de mapa de carreteras resuelve el problema de planificación en un tiempo exponencial con respecto a la dimensión del espacio de configuraciones, se basa en construir la silueta del espacio libre del robot, añadirle segmentos curvos que ligan "puntos críticos" de la misma con otros segmentos curvos. El mapa de carreteras se forma por la silueta y los segmentos curvos.

3.4.2 Descomposición de celdas

Los métodos de descomposicion de celdas se basan en descomponer el área libre del espacio de configuraciones en regiones. Hay dos formas de obtener las regiones: la descomposición exacta de celdas y la descomposición aproximada.

Descomposición exacta de celdas

El enfoque de descomposición exacta de celdas se basa en descomponer el espacio libre en una colección de regiones cuya intersección es vacía y su unión

es exactamente el área libre. Para encontrar un camino entre dos puntos se construye el grafo de conectividad que representa la adyacencia entre regiones, y a partir de ahí se realiza la búsqueda.

Si la búsqueda es exitosa, produce un canal formado por una serie de celdas adyacentes que contienen en sus extremos a las celdas con la configuración inicial y final. De esta serie se extrae la ruta que el robot debe seguir.

Un aspecto muy importante de este enfoque es la realización de las celdas, ya que se puede dar el caso de que al definirlas no sean lo mas adecuado para encontrar un camino. Por ejemplo si se decide hacer que el espacio entero del área libre se "descomponga" en una celda simple, no es sencillo realizar la planificación. Por esto se considera que las celdas generadas deben cumplir al menos con las siguientes características:

- La geometría de cada celda debe ser lo suficientemente simple como para que sea sencillo encontrar una ruta entre dos configuraciones en la misma celda.
- Debe ser fácil evaluar la adyacencia entre cualesquiera par de celdas y encontrar una ruta que cruce la frontera entre dos celdas adyacentes.

Cuando el espacio de configuraciones es bidimensional y los obstáculos son poligonales, se puede descomponer el espacio libre en trapezoides o tambien se puede modelar al robot como un segmento de línea que se traslada y gira entre obstáculos poligonales. En el último caso el espacio de configuraciones es de tres dimensiones y los obstáculos son regiones de tres dimensiones.

Un ejemplo de descomposición exacta de celdas se puede ver en la figura 3.6, en el cual el espacio libre se descompone en espacios de los cuales se puede hacer un grafo de adyacencias, y la búsqueda de un camino se reduce a una búsqueda de adyacencias.

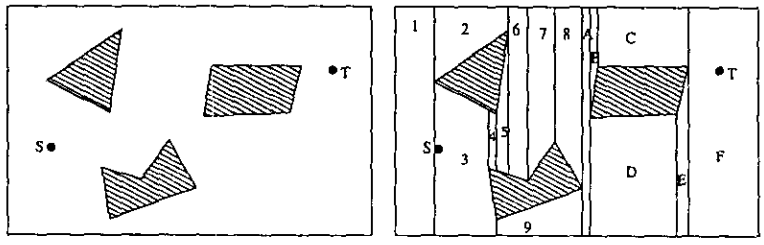


Figura 3.6: Descomposición exacta de celdas

Descomposición aproximada de celdas

La descomposición aproximada de celdas consiste en representar el espacio libre en celdas de una geometría rectangular que pueden no reflejar con exactitud el espacio libre, aunque se obtiene una aproximación conservativa del espacio libre.

De la misma manera que en la descomposición exacta de celdas se realiza un grafo de conectividad para representar la adyacencia entre regiones y se realiza la planificación de movimientos a partir del grafo.

Las ventajas de este enfoque sobre el de descomposición exacta son que se simplifica la descomposición, ya que se obtiene de iteraciones sobre el mismo algoritmo, y que los resultados no son sensibles a aproximaciones numéricas de cómputo. Como consecuencia, la descomposición aproximada es más sencilla de implantar que los métodos exactos. Además, se puede controlar la cantidad de espacio libre alrededor de los objetos fijando la dimensión mínima de las celdas.

El costo de no obtener una caracterización exacta del espacio libre es que se puede fallar para encontrar un camino, aún cuando exista, y que se pierde la caracterización de discontinuidades en las restricciones de movimientos.

El desempeño de este método es muy bueno cuando la dimensión es pequeña, por ejemplo, en el caso de un robot que se mueve sonbre un plano con obstáculos poligonales, pero el grado de complejidad de las búsquedas crece exponencialmente conforme la dimensión del espacio de configuraciones aumenta.

La descomposición se realiza, como se aprecia en la figura 3.7 iterativamente partiendo de una celda cuya área es igual a la del espacio de configuraciones, ésta se divide en partes iguales. Se verifica si las áreas resultantes quedan completamente libres de obstáculos, si no es así, se les aplica el mismo mecanismo. Este proceso se repite hasta que se llegue a áreas de un tamaño previamente determinado. Las áreas que quedaron totalmente libres de obstáculos se usan para hacer el grafo de conectividad.

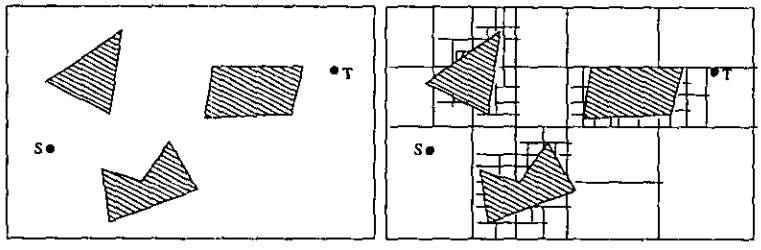


Figura 3.7: Descomposición aproximada de celdas

3.4.3 Métodos de campos potenciales

A diferencia de los métodos descritos con anterioridad, los métodos de campos potenciales no tratan de obtener la conectividad que hay en el espacio libre del robot. Ahora la idea es representar al robot como una partícula que sufre la influencia de un campo potencial artificial U cuyas variaciones locales reflejan la estructura del espacio libre y dependen de los obstáculos del entorno y del punto al que se dirije el robot [11].

Originalmente, el método de campos potenciales se utilizó con fines de navegación, para evitar obstáculos cuando no se tiene un modelo previo de los

mismos tratando de generar movimientos eficientes, más que para llegar a la meta. De hecho y debido al mecanismo de optimización que utilizan, es probable caer en un mínimo local distinto a la configuración meta. De cualquier modo, se puede combinar la técnica de campos potenciales con técnicas de búsqueda en grafos para realizar planificación de movimientos.

Para tratar con los mínimos locales se debe intentar definir una función potencial que no los tenga o que tenga muy pocos, además se pueden agregar al método técnicas para escapar de ellos.

La función potencial se define de manera que su valor en determinada configuración no dependa de la forma o de la distribución de los obstáculos mas allá de una distancia alrededor de la configuración, por ésto también se les conoce como métodos locales, a pesar de que para su definición se considera información que puede ser global, como la configuración meta.

La principal desventaja de los métodos de campos potenciales es que pueden fallar y no encontrar una ruta, aún cuando ésta exista, pero por otro lado, son muy rápidos para llegar a una solución. De esta manera es posible construir un planificador eficiente y razonablemente confiable.

Típicamente, la función potencial sobre el espacio libre se define como la suma de un potencial de atracción que empuja al robot hacia la configuración meta y de un potencial repulsivo que lo aleja de los obstáculos. La planificación de movimientos se hace a través de iteraciones, en cada una de las cuales se induce una fuerza artificial

$$\vec{F}(q) = -\vec{\nabla}(q) \tag{3.1}$$

que obliga al robot a desplazarse en la dirección en que decrece el campo potencial.

El campo potecial se genera mediante la suma del campo atractivo U_{atr} con el campo repulsivo U_{rep}

$$U(q) = U_{atr}(q) + U_{rep}(q)$$

de lo que

$$\vec{F}(q) = \vec{F}_{atr}(q) + \vec{F}_{rep}(q)$$

donde

$$\vec{F}_{atr}(q) = -\vec{\nabla}_{atr}(q)$$
$$\vec{F}_{rep}(q) = -\vec{\nabla}_{rep}(q)$$

Campos potenciales repulsivos

El objetivo de los campos potenciales repulsivos es crear una fuerza que aleje al robot de los obstáculos, ésto se logra usando un potencial de un valor que tienda a infinito en la superficie de cada obstáculo y que se reduzca en puntos alejados del mismo. Un campo así puede ser

$$U_{rep}(q) = \frac{1}{2}\eta\frac{1}{\|q - q_{obst}\|^2} \tag{3.2}$$

Se puede establecer una distancia mínima de influencia d_0 para evitar que un objeto lejano al robot contribuya innecesariamente a crear una fuerza de repulsión. Modificando 3.2 de acuerdo a este criterio se obtiene:

$$U_{rep}(q) = \begin{cases} \frac{1}{2}\eta\left(\frac{1}{\|q-q_{obst}\|} - \frac{1}{d_0}\right)^2 & \text{si } \|q-q_{obst}\| \leq d_0 \\ 0 & \text{si } \|q-q_{obst}\| > d_0 \end{cases} \tag{3.3}$$

Derivando para encontrar la fuerza repulsiva se obtiene:

$$\vec{F}_{rep}(q) = \begin{cases} \eta\left(\frac{1}{\|q-q_{obst}\|} - \frac{1}{d_0}\right)\left(\frac{1}{\|q-q_{obst}\|^2}\frac{(q-q_{obst})}{\|q-q_{obst}\|}\right) & \text{si } \|q-q_{obst}\| \leq d_0 \\ 0 & \text{si } \|q-q_{obst}\| > d_0 \end{cases} \tag{3.4}$$

Siempre que haya varios obstáculos, el campo potencial total es la superposición de los creados por cada obstáculo

$$U_{rep}(q) = \sum_{i=1}^{k} U_{rep}^k(q) \qquad \text{donde } k = \text{Número del obstáculo}$$

Campos potenciales atractivos

Los campos potenciales atractivos crean una fuerza de atracción hacia la configuración meta del robot. Se puede considerar un campo de tipo parabólico de la forma

$$U_{atr}(q) = \frac{1}{2}\varepsilon_1\|q - q_{dest}\|^2$$

Derivando para obtener la fuerza de atracción se obtiene

$$\vec{F}_{atr}(q) = -\varepsilon_1\|q - q_{dest}\|$$

este campo presenta valores de fuerzas elevados para puntos lejanos al destino, lo que tiene como consecuencia aceleraciones grandes, ya que

$$F_{atr}(q) \to \infty \text{ cuando } \|q - q_{dest}\| \to \infty$$

Para mantener acotadas las fuerzas de atracción, se puede considerar al campo atractivo de tipo cónico

$$U_{atr}(q) = \varepsilon_2\|q - q_{dest}\|$$

de lo que

$$\vec{F}_{atr}(q) = \varepsilon_2\frac{(q - q_{dest})}{\|q - q_{dest}\|}$$

en donde para $q = (x, y, z)^T$

$$\vec{\nabla}U = \begin{pmatrix} \frac{\partial U}{\partial x} & \frac{\partial U}{\partial y} & \frac{\partial U}{\partial z} \end{pmatrix}^T$$

en este caso la amplitud de la fuerza atractiva es constante, aún acercándose al destino con la posibilidad de generar inestabilidades, lo cual tampoco es deseable.

Como ya se dijo, los campos parabólicos no convienen cuando la distancia al destino es grande debido a que las aceleraciones generadas son muy altas. Por otro lado los campos cónicos presentan el inconveniente de mantener fuerzas atractivas constantes, aùn a distancias muy cercanas al destino. Para resolver estos inconvenientes se utiliza una combinación de ambos como se muestra a continuación:

$$U_{atr}(q) = \begin{cases} \frac{1}{2}\varepsilon_1 \|q - q_{dest}\|^2 & \text{si } \|q - q_{dest}\| \leq d_0 \\ \varepsilon_2 \|q - q_{dest}\| & \text{si } \|q - q_{dest}\| > d_0 \end{cases} \tag{3.5}$$

En donde, para que la función sea contínua se debe cumplir con

$$\frac{1}{2}\varepsilon_1 d_0 = \varepsilon_2$$

Hay situaciones en las que las fuerzas se equilibran aún cuando el robot no ha llegado a la meta establecida. Ésto pasa principalmente cuando el robot se encuentra muy cerca de la meta y en los alrededores de ésta hay un objeto, de manera que la fuerza de atracción es igual a la de repulsión. Una forma de resolver ésta situación es aplicando una fuerza mínima adicional a la fuerza de atracción, que sea superior a la fuerza de repulsión del objeto que rechaza al robot.

Planificación

Hay varios métodos de planificación utilizando campos potenciales, aquí se presentan dos de los más representativos. El primero consiste en generar una trayectoria a partir de una secuencia de segmentos de recta entre la configuración inicial del robot y la meta. La longitud y dirección de cada segmento son la magnitud y la dirección del vector de fuerzas resultante en el punto que está el robot, una vez que se ha seguido el segmento de recta encontrado, se calcula otro, y se itera de esta manera hasta alcanzar la configuración meta.

En el segundo método, se utiliza el vector gradiente para guiar directamente el desplazamiento. Para ello se define un vector unitario en la dirección del gradiente

$$\vec{f}(q) = \frac{\vec{F}(q)}{\|\vec{F}(q)\|}$$

de manera que el desplazamiento en instantes discretos se define mediante

$$q_{i+1} = q_i + \delta_i \vec{f}(q) \tag{3.6}$$

donde el valor de δ_i se puede establecer de una de las siguientes maneras:

- constante durante todo el recorrido;

- variable, dependiento de la proximidad a los obstáculos y al destino; o
- directamente proporcional a la fuerza resultante en cada punto.

El módulo del segmento de recta entre q_{i-1} y q_i, debe estar acotado superiormente por:

- La distancia al obstáculo más próximo, para garantizar que no haya colisión con éste.
- La distancia al destino, con lo que se evita llegar mas allá del mismo.

Por ejemplo, utilizando una función parabólica de dos dimensiones para mostrar que después de iterar se llega siempre al mínimo de la función, se puede usar

$$y = y_0 + (x - x_0)^2$$

que representa el campo potencial total $U(x) = y$ como se ve en la figura 3.8.

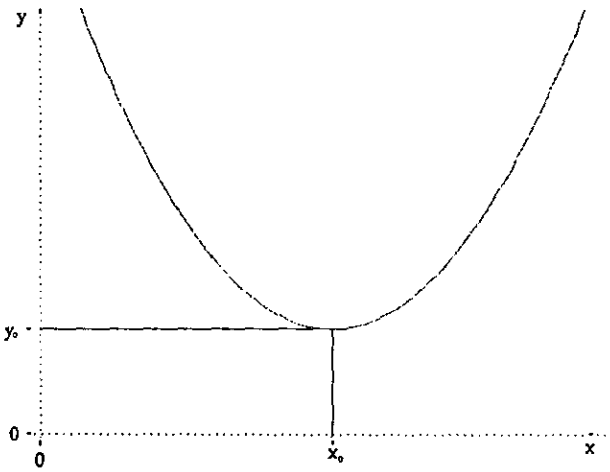


Figura 3.8: Función de ejemplo para un campo potencial

Derivando se obtiene

$$\nabla U(x) = \frac{dy}{dx} = 2(x - x_0)$$

A partir de ésto y utilizando las ecuaciones 3.6 y 3.1 se obtiene que

$$x_n = x_{n-1} - \delta_{n-1}\frac{dy}{dx}$$

por lo que

$$x_n = x_{n-1} - 2\delta_{n-1}x_{n-1} + 2\delta_{n-1}x_0$$

Dependiendo del valor de δ_{n-1} la función se aproxima más rápido o lento al destino. Si para este ejemplo se utiliza una

$$\delta_{n-1} = \frac{1}{2}$$

que es constante en todas las iteraciones, se concluye que $x_n = x_0$, y en x_0

$$\frac{dy}{dx} = 0$$

por lo que a pendiente es 0 y se llega al destino en un paso.

En la figura 3.9 se puede ver una ruta generada utilizando el método de campos potenciales. En éste caso la estructura que sobresale representa un obstáculo que genera un campo repulsivo con una función cuadrática inversa. Por su parte el punto meta genera un campo atractivo de forma parabólica en sus cercanias y de forma lineal en regiones mas alejadas, es por ésto que se observa una inclinación en toda la superficie generada.

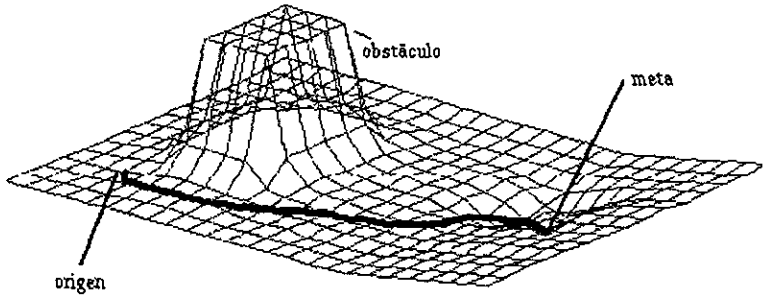


Figura 3.9: Ruta encontrada usando campos potenciales

3.5 Restricciones cinemáticas

En las secciones anteriores se han descrito métodos de planificación de rutas considerando al robot como un objeto que se puede mover libremente, sin embargo en la realidad los robots están sujetos a restricciones cinemáticas que les impiden trasladarse y rotar con libertad. Por ejemplo, si un robot está compuesto por eslabones articulados, éstos le impiden realizar con libertad algunos movimientos. Otro caso es el de un robot móvil con estructura similar a la de un carro, aunque se puede desplazar y girar, no se puede mover hacia los lados, y los giros que realiza están limitados por el rango de su volante.

Las restricciones que afectan a un robot articulado se pueden representar mediante ecuaciones que relacionan los parámetros de configuración y que se pueden usar para eliminar algunos parámetros y reducir la dimensión del espacio de configuraciones. A estas ecuaciones se les llama *restricciones holonómicas.*

La planificación de rutas cuando el robot está sujeto a restricciones holonómicas es prácticamente igual que la que se realiza en un robot "libre". El efecto de estas restricciones afecta la construcción del espacio de configuraciones, e incluso en ocasiones lo simplifica. Por ejemplo, si se sabe que un robot tendrá orientación constante, no es necesario incluir en el espacio de configuraciones todas las posibles orientaciones para él y el espacio reduce su dimensión en uno. La mayoría de las veces las restricciones holonómicas que se expresan en una ecuación reducen la dimensión del espacio, y las inecuaciones eliminan parámetros.

Las restricciones que afectan los movimientos de un robot tipo coche se pueden convertir en una ecuación y una desigualdad que involucran los parámetros de velocidad del robot, reduciendo la dimensión del espacio de velocidades del mismo. A éstas se les llama *restricciones no holonómicas.*

Una técnica de planificación de movimientos en robots sujetos a restricciones no holonómicas construye primero un camino libre sin considerarlas y lo transforma en un camino libre factible topológicamente, en la que se incluyen los movimientos necesarios para poder llegar a las configuraciones parciales. El problema es que se pueden generar demasiados movimientos. Existen otras técnicas que han sido diseñadas específicamente para tipos específicos de robot.

3.6 Navegación

Como se dijo al principio del capítulo, el navegador se encarga de encontrar un movimiento contínuo que lleve al robot de una posición a otra a través de una ruta geométrica localmente definida.

3.6.1 Modelo matemático

La forma mas simple de llevar al robot de la configuración q_A a la configuración q_B es dividir la ruta en una secuencia de lineas rectas, como se observa en la figura 3.10.

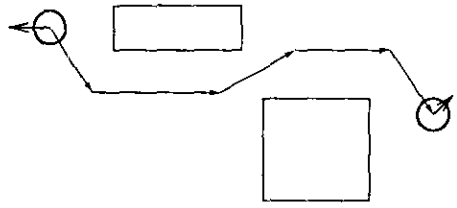


Figura 3.10: Ruta descompuesta en una secuencia de rectas

El navegador recibe esa secuencia del planificador y genera configuraciones que el robot pueda alcanzar con movimientos simples, es decir, giros y desplazamientos. Cada vez que el robot realiza un giro y un desplazamiento, el

robot pasa de la configuración q_i a la configuración q_{i+1}. La posición de q_{i+1} es el siguiente punto a alcanzar moviendose en línea recta, y su orientación es la orientación necesaria para que el robot apunte hacia el siguiente punto. En resumen, el robot primero se orienta hacia el siguiente punto y después avanza para alcanzarlo.

Como se mencionó en la sección 3.2, el robot está ligado a un plano F_A y como consecuencia, tanto su posición como su orientación están en función de la posición y orientación de F_A con respecto a F_W, que es el plano ligado al espacio de trabajo del robot.

Si se asume que el robot sólo se puede mover en la dirección del eje x de F_A y que puede girar sobre su eje para orientarse en cualquier dirección, sólo hace falta encontrar las ecuaciones que rigen sus giros y desplazamientos.

La figura 3.11 muestra las configuraciones q_i y q_{i+1}. Se puede observar que para que el robot pase de la configuración q_i a q_{i+1}, primero debe orientarse de manera que su eje x_A apunte hacia el origen de q_{i+1} ($O_{q_{i+1}}$), es decir, debe adquirir la orientación θ_{i+1} y después moverse sobre el segmento de recta $\overline{O_{q_i}O_{q_{i+1}}}$.

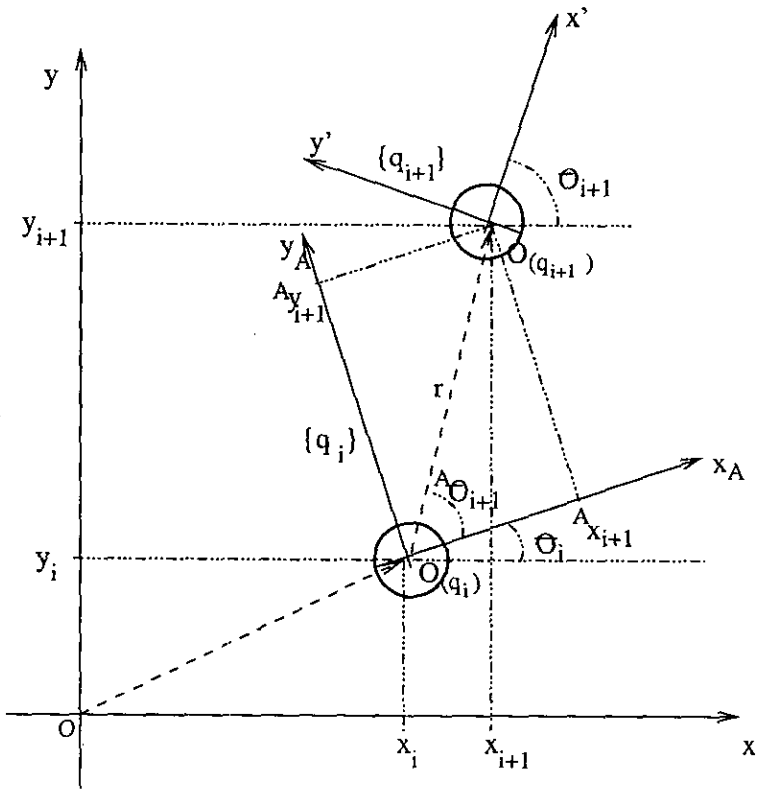


Figura 3.11: Sistemas de referencia

El concepto de transformación [12] nos ayuda a encontrar los movimientos que debe realizar el robot, localizando la posición y orientación del robot en la configuración q_{i+1} con respecto a la configuración q_i. En la figura 3.11 se observa que

$$\begin{pmatrix} x_{i+1} \\ y_{i+1} \end{pmatrix} = \overline{O_{q_i}} + \overline{O_{q_{i+1}}} = \begin{pmatrix} x_i \\ y_i \end{pmatrix} + \begin{pmatrix} r\cos({}^A\theta_{i+1} + \theta_i) \\ r\sin({}^A\theta_{i+1} + \theta_i) \end{pmatrix}$$

en donde

$${}^A\theta_{i+1} = \theta_{i+1} - \theta_i \tag{3.7}$$

Aplicando identidades trigonométricas queda:

$$\begin{pmatrix} x_{i+1} \\ y_{i+1} \end{pmatrix} = \begin{pmatrix} x_i + r\cos^A\theta_{i+1}\cos\theta_i - r\sin^A\theta_{i+1}\sin\theta_i \\ y_i + r\cos^A\theta_{i+1}\sin\theta_i + r\sin^A\theta_{i+1}\cos\theta_i \end{pmatrix} \tag{3.8}$$

Por otro lado:

$$\begin{pmatrix} {}^Ax_{i+1} \\ {}^Ay_{i+1} \end{pmatrix} = \begin{pmatrix} r\cos^A\theta_{i+1} \\ r\sin^A\theta_{i+1} \end{pmatrix} \tag{3.9}$$

Sustituyendo 3.9 en 3.8

$$\begin{pmatrix} x_{i+1} \\ y_{i+1} \end{pmatrix} = \begin{pmatrix} x_i + {}^Ax_{i+1}\cos\theta_i - {}^Ay_{i+1}\sin\theta_i \\ y_i + {}^Ax_{i+1}\sin\theta_i + {}^Ay_{i+1}\cos\theta_i \end{pmatrix}$$

de lo cual:

$$\begin{pmatrix} x_{i+1} \\ y_{i+1} \end{pmatrix} = \begin{pmatrix} \cos\theta_i & -\sin\theta_i \\ \sin\theta_i & \cos\theta_i \end{pmatrix} \begin{pmatrix} {}^Ax_{i+1} \\ {}^Ay_{i+1} \end{pmatrix} + \begin{pmatrix} x_i \\ y_i \end{pmatrix}$$

Aumentando ésta ecuación de manera que incluya el tercer parámetro de la configuración que es la orientación y se obtiene de 3.7 queda:

$$\begin{pmatrix} x_{i+1} \\ y_{i+1} \\ \theta_{i+1} \end{pmatrix} = \begin{pmatrix} \cos\theta_i & -\sin\theta_i & 0 \\ \sin\theta_i & \cos\theta_i & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} {}^Ax_{i+1} \\ {}^Ay_{i+1} \\ {}^A\theta_{i+1} \end{pmatrix} + \begin{pmatrix} x_i \\ y_i \\ \theta_i \end{pmatrix} \tag{3.10}$$

que también se puede expresar como

$$q_{i+1} =_i R^A q_{i+1} + q_i$$

en donde ${}_iR$ es la matriz de senos y cosenos que representa la rotación de la configuración q_i. Esta matriz es ortonormal, por lo que

$$R^{-1} = R^T$$

despejando se obtiene

$${}^Aq_{i+1} =_i R^{-1}(q_{i+1} - q_i) =_i R^T(q_{i+1} - q_i) \tag{3.11}$$

conceptualmente

$$ {}^{A}q_{i+1} = \begin{pmatrix} {}^{A}x_{i+1} \\ {}^{A}y_{i+1} \\ {}^{A}\theta_{i+1} \end{pmatrix} $$

es el avance del robot sobre el nuevo eje de coordenadas en x e y.

Finalmente, sustituyendo 3.11 en 3.10 se obtiene la configuración q_{i+1} a partir del robot A:

$$ \begin{pmatrix} {}^{A}x_{i+1} \\ {}^{A}y_{i+1} \\ {}^{A}\theta_{i+1} \end{pmatrix} = \begin{pmatrix} \cos\theta_i & \sin\theta_i & 0 \\ -\sin\theta_i & \cos\theta_i & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_{i+1} - x_i \\ y_{i+1} - y_i \\ \theta_{i+1} - \theta_i \end{pmatrix} \tag{3.12} $$

De aquí se concluye que el robot primero tiene que girar ${}^{A}\theta_{i+1}$ unidades, y después desplazarse ${}^{A}x_{i+1}$ unidades (ya que sólo se mueve en la dirección del eje X^{+}).

$$ {}^{A}\theta_{i+1} = \theta_{i+1} - \theta_i \tag{3.13} $$

$$ {}^{A}x_{i+1} = (x_{i+1} - x_i)\cos\theta_i + (y_{i+1} - y_i)\sin\theta_i \tag{3.14} $$

Para ilustrar el uso de las expresiones 3.13 y 3.14 se plantea el problema de encontrar los movimientos necesarios para seguir la secuencia mostrada en la figura 3.12.

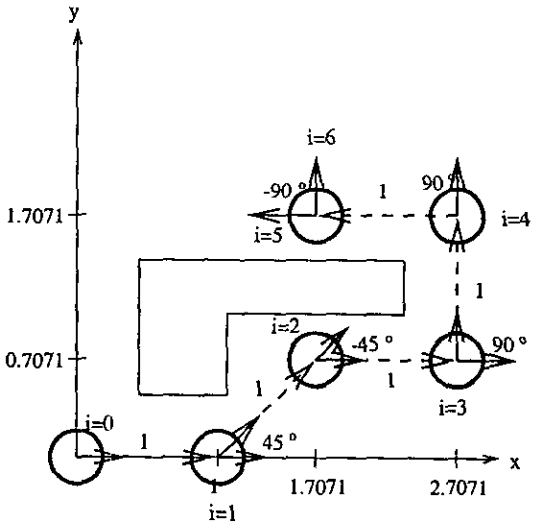


Figura 3.12: Ejemplo de navegación

En $i = 0$ el robot se encuentra en el origen del sistema ($x_0 = 0$, $y_0 = 0$) con un ángulo de 0^{o} con respecto al eje x ($\theta_0 = 0$). Para llegar a la configuración

en la que $x_1 = 1$, $y_1 = 0$ y $\theta_1 = 0$ se utilizan las ecuaciones 3.13 y 3.14 para encontrar el desplazamiento y giro necesarios:

$$^A\theta_1 = 0^o - 0^o = 0^o$$

$$^Ax_1 = (1-0)\cos 0^o + (0-0)\sin 0^o = 1$$

$$^Ay_1 = -(1-0)\sin 0^o + (0-0)\cos 0^o = 0$$

Ahora $i = 1$ y la meta es llegar a la configuración $x_2 = 1 + \frac{1}{\sqrt{2}}$, $y_2 = \frac{1}{\sqrt{2}}$ y $\theta_2 = 45^o$, se vuelven a aplicar las fórmulas:

$$^A\theta_2 = 45^o - 0^o = 45^o$$

$$^Ax_2 = (1 + \frac{1}{\sqrt{2}} - 1)\cos 45^o + (\frac{1}{\sqrt{2}} - 0)\sin 45^o = 1$$

$$^Ay_2 = -(1 + \frac{1}{\sqrt{2}} - 1)\sin 45^o + (\frac{1}{\sqrt{2}} - 0)cos45^o = 0$$

Para $i = 2$, $x_3 = 2 + \frac{1}{\sqrt{2}}$, $y_3 = \frac{1}{\sqrt{2}}$ y $\theta_3 = 0^o$, los movimientos son:

$$^A\theta_3 = 0^o - 45^o = -45^o$$

$$^Ax_3 = (2 + \frac{1}{\sqrt{2}} - 1 - \frac{1}{\sqrt{2}})\cos 0^o + (\frac{1}{\sqrt{2}} - \frac{1}{\sqrt{2}})\sin 0^o = 1$$

$$^Ay_3 = -(2 + \frac{1}{\sqrt{2}} - 1 - \frac{1}{\sqrt{2}})\sin 0^o + (\frac{1}{\sqrt{2}} - \frac{1}{\sqrt{2}})\cos 0^o = 0$$

Si $i = 3$, $x_4 = 2 + \frac{1}{\sqrt{2}}$, $y_4 = 1 + \frac{1}{\sqrt{2}}$ y $\theta_4 = 90^o$, entonces:

$$^A\theta_4 = 90^o - 0^o = 90^o$$

$$^Ax_4 = (2 + \frac{1}{\sqrt{2}} - 2 - \frac{1}{\sqrt{2}})\cos 90^o + (1 + \frac{1}{\sqrt{2}} - \frac{1}{\sqrt{2}})\sin 90^o = 1$$

$$^Ay_4 = -(2 + \frac{1}{\sqrt{2}} - 2 - \frac{1}{\sqrt{2}})\sin 90^o + (1 + \frac{1}{\sqrt{2}} - \frac{1}{\sqrt{2}})\cos 90^o = 0$$

Cuando $i = 4$, $x_5 = 1 + \frac{1}{\sqrt{2}}$, $y_5 = 1 + \frac{1}{\sqrt{2}}$ y $\theta_5 = 180^o$, se tiene que:

$$^A\theta_5 = 180^o - 90^o = 90^o$$

$$^Ax_5 = (1 + \frac{1}{\sqrt{2}} - 2 - \frac{1}{\sqrt{2}})\cos 180^o + (1 + \frac{1}{\sqrt{2}} - 1 - \frac{1}{\sqrt{2}})\sin 180^o = 1$$

$$^Ay_5 = -(1 + \frac{1}{\sqrt{2}} - 2 - \frac{1}{\sqrt{2}})\sin 180^o + (1 + \frac{1}{\sqrt{2}} - 1 - \frac{1}{\sqrt{2}})\cos 180^o = 0$$

Finalmente, para $i = 5$, $x_6 = 1 + \frac{1}{\sqrt{2}}$, $y_6 = 1 + \frac{1}{\sqrt{2}}$ y $\theta_6 = 90^o$, los movimientos serán:

$$^A\theta_6 = 90^o - 180^o = -90^o$$

$$^{A}x_6 = (1 + \frac{1}{\sqrt{2}} - 1 - \frac{1}{\sqrt{2}})\cos 90^o + (1 + \frac{1}{\sqrt{2}} - 1 - \frac{1}{\sqrt{2}})\sin 90^o = 0$$

$$^{A}y_6 = -(1 + \frac{1}{\sqrt{2}} - 1 - \frac{1}{\sqrt{2}})\sin 90^o + (1 + \frac{1}{\sqrt{2}} - 1 - \frac{1}{\sqrt{2}})\cos 90^o = 0$$

llegando a la meta establecida en la figura 3.12. En la misma figura se observa junto a las líneas que sigue el robot la distancia recorrida, y el ángulo de giro se puede ver junto al robot, donde éste cambia de dirección. Cabe notar que los giros positivos son en el sentido opuesto a las manecillas del reloj.

i	x_i	y_i	θ_i	x_{i+1}	y_{i+1}	θ_{i+1}	$^{A}x_{i+1}$	$^{A}y_{i+1}$
0	0	0	0	1	0	0	1	0
1	1	0	0	1.707	.707	45	1	0
2	1.707	.707	45	2.707	.707	0	1	0
3	2.707	.707	0	2.707	1.707	90	1	0
4	2.707	1.707	90	1.707	1.707	180	1	0
5	1.707	1.707	180	1.707	1.707	90	0	0
6	1.707	1.707	90	*	*	*	*	*

Tabla 3.1: Ejemplo de movimientos generados por el navegador

En la tabla 3.1 se muestra el resumen de los datos usados y obtenidos en cada paso. Se puede ver que el valor de $^{A}y_{i+1}$ siempre es 0 y que la distancia entre la posición del robot y el punto al que se quiere desplazarse desplaza en el tiempo $i+1$ coincide con el valor de $^{A}x_{i+1}$, debido a que el robot siempre se mueve en la dirección de su eje x.

Las ecuaciones 3.13 y 3.14 se pueden aplicar en técnicas de navegación que cuentan con toda la información del entorno y también en las que no la tienen, pero cuando no se cuenta con información completa suelen ser insuficientes, así que es necesario realizar la navegación utilizando sensores y auxiliandose del piloto.

3.7 Pilotaje

El piloto es el encargado de seguir las trayectorias que le indica el navegador, en un tiempo previamente establecido. También es el encargado de evadir los objetos que se puedan encontrar y que el planificador no previó para generar la ruta.

Para evadir los obstáculos el piloto requiere las lecturas de los sensores del robot, entre los que destacan los sensores de proximidad y los de tacto. Dependiendo del tipo de sensores es la manera en que se realiza el pilotaje.

3.7.1 Pilotaje con sensores de tacto

La forma mas sencilla de sensado en un robot móvil es mediante sensores de tacto. El robot hace el recorrido de su posición inicial S a su posición objetivo

T a través de una linea recta hasta que llega a T o interactúa con el *i-esimo* obstáculo en el punto de contacto H_i. Si se dió el contacto el robot sigue el perímetro del obstáculo hasta que llega al punto de salida L_i y continua en línea recta hacia T, como se puede ver en la figura 3.13. Dado que no se conocen las características del objeto con el que se hizo contacto, la dirección en que se le rodea se establece de manera arbitraria, por ejemplo a la izquierda. El problema de seguir este algoritmo es que es probable que por la forma de los obstáculos o por la distribución de éstos en el entorno del robot, nunca se llegue al objetivo a pesar de haber un camino.

Para resolver este problema se han propuesto los algoritmos Bug1 y Bug2 [13] que asumen que el robot puede conocer en todo momento sus coordenadas y las de su objetivo.

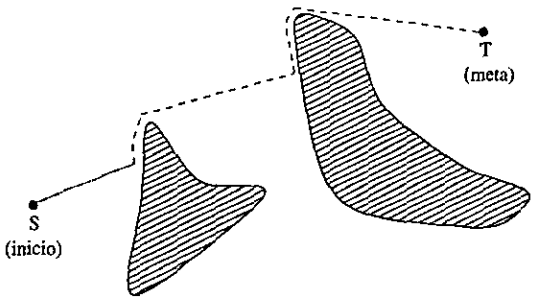


Figura 3.13: Navegación con sensores de tacto

El algoritmo Bug1

El algoritmo Bug1 consta de los siguientes pasos:

1. Del punto L_{i-1} (con $L_0 = S$, mueve al robot hacia T a lo largo de la recta (S,T) hasta que ocurra alguna de las siguientes opciones:
   (a) T es alcanzado; el proceso termina.
   (b) Se encuentra un obstáculo en el punto de contacto H_i; ve al paso 2.
2. Usando la dirección de movimiento predefinida, sigue la orilla del obstáculo. Si se llega a T, para. Mientras se avanza, se almacenan las coordenadas del punto actual en el registro R_1, el punto visitado mas cercano a T se almacena en Q_m, la distancia recorrida desde H_j en el registro R_2 y la distancia recorrida desde Q_m se guardan en el registro R_3. Después de haber rodeado completamente al obstáculo (se llegó de nuevo a H_i) se define el punto de salida $L_i = Q_m$ y se continúa con el paso 3.
3. Si hay un segmento de recta entre L_i y T que cruze el obstáculo en el punto L_i, termina el proceso, porque el objetivo no es alcanzable. En otro

caso utiliza R_2 y R_3 para determinar el camino más corto a L_i, se sigue el camino elegido y se establece $i = i + 1$ y se regresa a 1.

En la figura 3.14 se observa el ejemplo de un robot rodeando los obstáculos con el algoritmo Bug1.

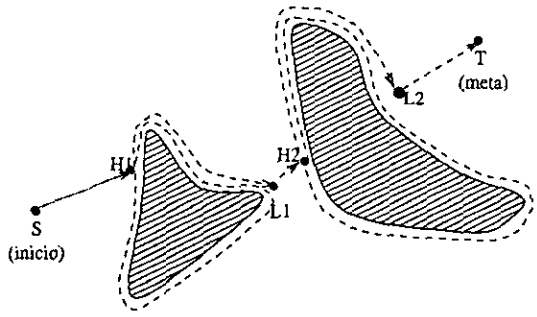


Figura 3.14: Ejemplo del algoritmo Bug1

3.7.2 El algoritmo Bug2

El algoritmo Bug2 es una variante de Bug1. En la figura 3.15 se observa un ejemplo de aplicar los pasos que se detallan a continuación:

1. Del punto L_{j-1} (con $L_0 = S$), mueve al robot hacia T a lo largo de la recta (S,T) hasta que ocurra una de las siguientes opciones:
   (a) T es alcanzado; termina el proceso.
   (b) Se encuentra un obstáculo en el punto de contacto H_j; se sigue en el paso 2.
2. Usando la dirección de movimiento predefinida, se sigue el contorno del obstáculo hasta que:
   (a) T es alcanzado; termina el proceso.
   (b) La recta (S,T) se encuentra en un punto Q tal que la distancia $D(Q,T) < D(H_j,T)$ y la recta (Q,T) no cruza al obstáculo en el punto Q. Se define el punto de salida $L_j = Q_m$, se establece $j = j+1$ y se va al paso 1.
   (c) El robot recorre totalmente el contorno del obstáculo y regresa a H_j sin haber definido el siguiente punto de contacto H_{j+1}, en este caso termina el proceso, ya que el objetivo está "atrapado" y es inalcanzable.

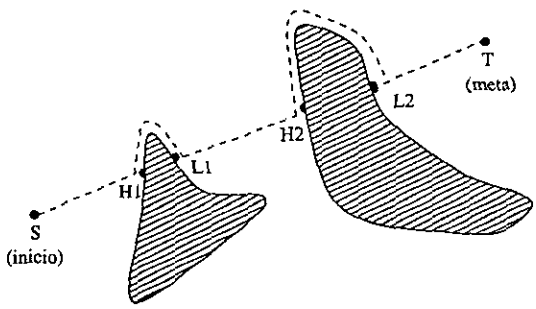


Figura 3.15: Ejemplo del algoritmo Bug2

En la práctica la eficiencia de estos dos algoritmos es variable, pero se puede decir que Bug1 es más conservador, ya que garantiza que el punto de salida es el más cercano a la meta. Bug2 es más agresivo y muchas veces más eficiente, ya que no requiere rodear todo el obstáculo.

3.7.3 Pilotaje con sensores de proximidad

Cuando se cuenta con sensores de proximidad, tales como el sonar, el sensor infrarrojo, sensores capacitivos, inductivos, de laser o visión, se pueden aprovechar sus características para mejorar el desempeño del robot. Por ejemplo se pueden modificar los algoritmos Bug1 y Bug2 para no necesitar regresar al punto de salida L_i definido, en lugar de ésto se puede "cortar" el camino y dirigirse a la meta en cuanto no haya obstáculo cercano "visible" por los sensores.

Cuando sólo se cuenta con información local y las decisiones de pilotaje se deben tomar dinámicamente, el robot debe ser capaz de decidir rápido [14]. El algoritmo para resolver este problema se puede dividir en: un algoritmo de selección de submeta (SSA) y un algoritmo de decisión de dirección (SDA). El SSA genera una submeta que indica una dirección temporal a seguir cuando la meta final no es visible y el SDA se genera una trayectoria de referencia que dirige al robot hasta que se genera la siguiente submeta.

En el SSA se pueden buscar las esquinas de los objetos que se tienen al alcanze de la vista y elegir como submeta aquella que esté más cerca de la meta. El SDA por su parte solamente genera las instrucciones necesarias para que el robot alcanze la submeta planteada, para esto puede utilizar el modelo matemático descrito en la sección 3.6.1

3.8 Navegación utilizando comportamientos

Para aprovechar el enfoque de control de robots basado en comportamientos usando una organización lógica como la descrita al principio de éste capítulo, es necesario realizar un módulo que al mismo tiempo sea navegador y piloto.

Debido a que se requieren al menos dos comportamientos "opinen" acerca de los movimientos que debe hacer el robot y un árbitro que decida acuál obedecer, se requiere que un sólo módulo cumpla con la misión que en el enfoque tradicional se delega a un navegador y un piloto independientes.

Una manera de implantar los comportamientos, es hacer uno que le dé al robot una orientación tal que se dirija a su meta, y otro que se encarge de detectar obstáculos y tratar de identificar configuraciones intermedias para evadirlo de manera similar al pilotaje con sensores de proximidad. La principal ventaja de éste enfoque es que resulta relativamente sencillo aumentarle funcionalidad al robot incorporando nuevos comportamientos que colaboren para hacer que el robot se desempeñe de una manera más "inteligente".

En ese contexto las misiones definidas para el planificador, el navegador y el piloto se trasladan. Una de las tareas del navegador, que es encontrar las distancias y ángulos de giro, se traslada al módulo de comportamientos, que también define velocidades y movimientos contínuos además de realizar las tareas del piloto.

3.9 Incertidumbres

En las secciones anteriores se presentaron métodos que se basan en consideraciones que en entornos reales son difíciles de encontrar. Por ejemplo, la mayoría de los objetos con los que interactuamos diariamente no son polihedros, además que identificar las características de un objeto a través de sensores no es sencillo.

Por otro lado se tiene al control del robot. Un robot móvil rara vez se desplaza en pisos lisos y perfectamente planos, por el contrario, sus movimientos se realizan en la mayoría de las veces en pisos rugosos. Para tener información acerca de la posición en la que se encuentra el robot, generalmente se confía en encodificadores que informan a un sistema sobre el número de veces que ha girado el eje de un motor. Esta información se utiliza para estimar la distancia que el robot se ha desplazado y la dimensión de sus giros, pero si los planes de movimientos se realizan asumiendo un piso liso, la información de los encoders no pasa de ser un referencia insuficiente para determinar en todo momento la posición del robot. Si además se fijan en la planificación las velocidades y aceleraciones de los motores como un aspecto crucial, el control se vuelve aún más complejo.

El problema básico de planificación tampoco involucra la posible presencia de objetos en movimiento. Si éstos son otros robots, se pueden involucrar en la planificación todos los robots que se piense estarán y considerarlos a todos un solo robot articulado, de esta manera los movimientos de todos los robots estarán coordinados. Pero muchas veces no es posible hacer ésto y se deben establecer mecanismos de comunicación. Si los objetos en movimiento no son controlables el problema se complica aún más, por ejemplo si de pronto se cruza enfrente del robot una persona, o se encuentra un objeto que no se había considerado en la planificación. Incluso si se hace una nueva planificación, el sistema se puede volver tan lento al grado de que no interese que un robot realize la tarea.

Por otro lado los algoritmos de navegación, en general, también asumen condiciones ideales, lo cuál como se dijo no sucede en la realidad.

El caso de la navegación basada en comportamientos, tiene la ventaja de que no es necesario plantear metas fijas y bien definidas al robot para que éste actúe. Por el contrario, una de las metas de éste enfoque es no utilizar datos "absolutos" para tomar decisiones, sin embargo todavía se deben realizar cálculos para que el sistema sea funcional.

Dada la gran cantidad de incertidumbres con que tiene que lidiar un robot y el tiempo de procesamiento que se consume en la planificación, se piensa que distribuirla daría mejores resultados que centralizarla, lo que justifica la existencia de enfoques de control como el de comportamientos.

Capítulo 4

Análisis

A partir de este capítulo se describe el proceso realizado para resolver el problema de implantar un robot capaz de desplazarse dentro de un entorno estático y conocido previamente.

4.1 Método de solución

Ya que se plantea hacer un controlador híbrido para el robot, también es necesario utilizar un método híbrido para implantarlo. El método utilizado se basa en el ciclo de vida del software y en la metodología BAT (Behavior Analysis and Training) [15] y en los métodos de desarrollo de Sistemas Expertos.

El ciclo de vida del software es un conjunto de pasos propuestos para desarrollar sistemas de información dentro de la Ingeniería de Software [16]. Está compuesto de cinco etapas:

- Análisis.
- Diseño.
- Implantación.
- Pruebas.
- Mantenimiento.

La metodología BAT fué propuesta por Marco Colombeti, Marco Dorigo y Giuseppe Borghi, como una metodología para Ingeniería de Comportamientos. Su principal propósito es ayudar a desarrollar robots basados en comportamientos que cumplan con las expectativas puestas en ellos. la metodología BAT se compone de las siguientes etapas:

- Descripción de la aplicación y requisitos del comportamiento del robot.
- Análisis de comportamientos.

- Especificación.
- Diseño, Implantación y verificación del robot naciente.
- Entrenamiento.
- Aseguramiento del comportamiento.

Los métodos de desarrollo de sistemas expertos son un tanto distintos de la Ingeniería de Software clásica. La principal razón es que éstos tratan de emular a un experto que para resolver un problema razona de una manera que es difícil describir en términos de algoritmos. El proceso de desarrollo involucra la realización de un conjunto de tareas que se repiten hasta que el sistema cumple con las especificaciones establecidas. A continuación se listan los pasos para desarrollar un Sistema Experto [17]:

- Definición de conocimientos.
- Diseño de conocimientos.
- Codificación.
- Verificación de conocimientos.
- Evaluación.

Después de adaptar estos métodos a las características del problema que se aborda en este trabajo, se descompuso el proceso en las siguientes etapas:

1. Descripción de la aplicación y requisitos del comportamiento del robot.
2. Análisis de comportamientos.
3. Especificación del entorno y del controlador.
4. Elección de la plataforma de desarrollo.
5. Diseño e implantación de los módulos para el control del robot.
6. Pruebas.

En la etapa de *descripción de la aplicación y requisitos del comportamiento del robot* se realiza una descripción concisa del robot y de su entorno inicial, también se describe el comportamiento global que se espera del robot.

En el *análisis de comportamientos* se genera una lista estructurada [15] de los comportamientos necesarios para cumplir con el comportamiento global, tomando en cuenta las características del robot y las de su entorno. Esta lista también contempla la relación que puede existir entre los comportamientos.

En la *especificación del entorno y del controlador* se determina como utilizar los sensores y actuadores del robot y se describe el entorno extendido. El

entorno extendido es el entorno del robot con las modificaciones que se consideren necesarias para que pueda cumplir con su misión. También se define la arquitectura del controlador y una estrategia de entrenamiento.

En la *elección de la plataforma de desarrollo* se definen el software y el hardware a utilizar durante la implantación.

En el *diseño e implantación de los módulos para el control del robot* se describe la operación de cada uno de los módulos que conforman al controlador y se implantan por separado. Se define un mecanismo de comunicaciones para que operen en forma conjunta y se integran. En los módulos que se implanten utilizando sistemas expertos se sigue el método descrito para desarrollarlos, es decir: definición, diseño, codificación y verificación de conocimientos.

En la etapa de *pruebas* se verifica que los componentes tengan el funcionamiento esperado al operar de manera independiente y que el robot tenga el comportamiento global esperado al integrarlos.

En el resto de éste capítulo se tratan los primeros 4 puntos del método delineado, para los últimos dos puntos se destinan los próximos dos capítulos (uno para cada punto).

4.2 Descripción de la aplicación y requerimientos de comportamientos

El robot debe ser capaz de llegar a un punto solicitado por el usuario dentro de un entorno estático, usando un mapa que contiene la descripción del lugar. Es probable que mientras se mueve, el robot encuentre algún objeto estático que no estaba previsto en el mapa, ante esta situación, el robot se debe detener y si puede, evadirlo.

Se propuso un controlador que aprovecha las ventajas del "enfoque tradicional" y del "enfoque de comportamientos" previamente descritos. Sus componentes principales son: Un planificador, un navegador, un piloto, un controlador y una red neuronal.

El *planificador* genera una ruta que permite que el robot llegue a su meta sin chocar con los obstáculos predefinidos en un mapa. Debe generar rutas globales en un entorno compuesto por áreas grandes y rutas locales.

El *navegador* divide la ruta en una secuencia de configuraciones que le envía al piloto.

El *piloto* recibe del navegador la siguiente configuración y trata de llegar a ella sin chocar. Ésto se logró construyendo al piloto de manera que tiene dos comportamientos: uno que trata de posicionar al robot en la configuración deseada, y otro que evita colisiones con los obstáculos.

La *red neuronal* filtra las lecturas de los sonares del robot para tratar de determinar donde hay un obstáculo y ayudar a evitar colisiones.

El *controlador* controla directamente a los motores del robot y toma lecturas de los sensores.

Es necesario poner énfasis en que se está realizando un controlador híbrido

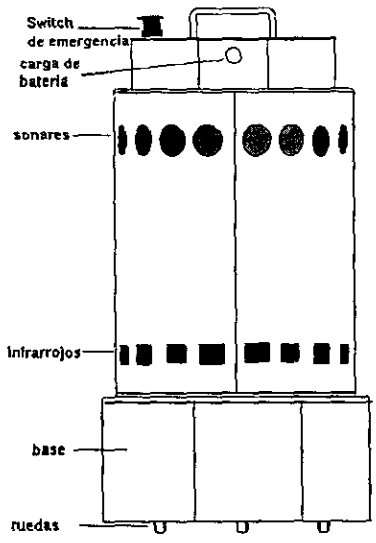


Figura 4.1: Robot B14 de RWI

que utiliza el enfoque tradicional en la estructura de capas del controlador, pero que en el piloto está usando el enfoque de comportamientos, con ayuda del controlador de bajo nivel del B14.

4.2.1 Descripción del robot

En la tabla 4.1 se listan las características del B14 y en la figura 4.1 se pueden ver algunos de sus componentes.

4.2.2 Descripción del entorno inicial

El entorno inicial del robot es el Laboratorio de Interfaces Inteligentes (figura 4.2) que está en la planta baja del edificio Luis G. Valdés Vallejo de la Facultad de Ingeniería en Ciudad Universitaria.

La iluminación del laboratorio es mixta, hay un ventanal muy grande que permite la entrada de luz solar y también recibe iluminación artificial de seis lámparas fluorescentes.

En el interior se encuentran distribuidas varias mesas metálicas fijas al piso, además de otros objetos que pueden cambiarse de lugar, tales como una caja de madera, varias sillas tubulares y otras cajas de cartón. Todos los objetos permanecen fijos mientras el robot se mueve.

Característica	*Comentario*
Forma de huella	Circular.
Diámetro	35 cm.
Altura	60 cm.
Peso	27.2 kg.
Distancia base-piso	1.27 cm.
Material	Aluminio anodizado.
Sensores	
Sonares	Anillo horizontal de 16 sonares a 50 cm.
Infrarrojos	Anillo horizontal de 16 emisor-sensor a 24 cm.
Tacto	22 sensores de tacto en toda su superficie.
Odómetro	Encodificadores en todos los actuadores, para conocer posición y orientación del robot.
Computadoras	
NEC 78310L	Microcontrolador para control de la base.
MC68HC11	Dos microcontroladores para controlar sensores.
Pentium	Computadora con SO Linux, tarjeta de red ethernet 10baseT y 10base2 y puerto serie.
Actuadores	
Motores	2 servo motores de 12V de DC de gran torque.
Giros	3 ruedas de giro.
Conducción	3 ruedas en arreglo sincrónico.
Diámetro de ruedas	8.26cm
Radio de giro	0
Velocidad traslación	90 cm/s.
Resolución traslación	1 mm.
Velocidad rotación	155°/s.
Resolución rotación	.35°.
Capacidad de carga	20 kg.
Energía	
Convertidor DC-DC	Suministra 10A @ +5V, 2A @ +12V y 2A @ -12V.
Baterías	2 intercambiables (hot-swappable), 192 W-hr.
Tiempo de recarga	4 horas apagado; 10 horas encendido.
Tiempo de autonomía	1 a 2 horas con movimientos, 3.5 horas inmovil.

Tabla 4.1: Características del B14 de Real World Interface

4.2.3 Requerimientos del comportamiento global

El usuario se comunica con el robot mediante una terminal o una computadora con conexión de red, usando el teclado o una interfaz gráfica. El robot puede recibir la orden de alcanzar una configuración (x, y, θ).

El planificador busca en un mapa una ruta óptima libre de colisiones, si la encuentra, le comunica al navegador la ruta para que el robot cumpla con la orden dada.

El navegador divide la ruta en una secuencia de trayectorias y transmite al piloto la siguiente configuración en la trayectoria, para que éste la alcanze.

Los comportamientos *alcanzar configuración* y *evadir obstáculo* realizan las funciones del piloto generando dos tipos de instrucciones para los controladores de los motores: giros y desplazamientos.

El comportamiento de *alcanzar configuración* es el encargado de llegar a la configuración indicada por el navegador. Para ello utiliza el odómetro y las ecuaciones que se trataron en la sección de *navegación* del capítulo anterior.

El comportamiento de *evadir obstáculo* busca obstáculos con los sonares y utiliza a la red neuronal para determinar el ángulo que tiene que girar el robot para que el obstáculo no le estorbe. A partir de esta información y de consultarle al planificador genera una configuración parcial para rodear el obstáculo.

Se puede ver que se está encargando al piloto una tarea que normalmente compete al navegador, ésto se hace porque se está utilizando el enfoque de comportamientos, y resulta más eficiente permitir que el piloto haga sus cálculos de giros y desplazamientos al mismo tiempo que el comportamiento *evade obstá-*

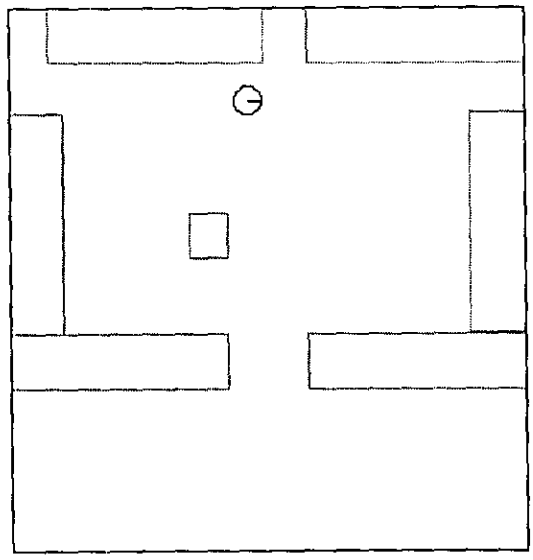


Figura 4.2: Vista superior del Laboratorio de Interfaces Inteligentes

culo genera sus propias propuestas de giros y desplazamientos. Si no se hiciera así, el piloto tendría que volver a calcular la posición que le solicita el navegador en base al ángulo de giro y el desplazamiento para poder evitar un obstáculo y alcanzar el objetivo.

4.3 Análisis de comportamientos

En esta sección se determinan los comportamientos necesarios para que el robot tenga el comportamiento global que resuelva el problema planteado. Aunque, como ya se dijo, el control del robot se realiza mezclando el enfoque tradicional y el de comportamientos, se puede hacer una descripción de todos los componentes del controlador en términos del comportamiento que se espera de ellos.

El comportamiento estructurado del sistema completo se puede describir de la siguiente manera:

*obedecer orden = (admitir orden · planificar ruta · seguir ruta)**

Esto significa que para obedecer una orden, el robot admite una orden, planifica una ruta y la sigue hasta llegar a la configuración meta. Una vez hecho ésto se repite el ciclo de manera repetitiva.

El comportamiento de *seguir ruta* se consiste en:

*seguir ruta = (generar configuración · llegar a configuración)**

Para seguir una ruta es necesario primero generar una configuración y luego llegar a ella. Este proceso se repite mientras haya configuraciones a alcanzar. El navegador es el responsable del comportamiento *generar configuración.*

El piloto es el que se encarga de mantener el comportamiento de *llegar a configuración* que está compuesto por dos comportamientos independientes que se activan para alcanzar las configuraciones parciales entre la posición inicial del robot y la configuración objetivo, como se ve a continuación:

*llegar a configuración= (alcanzar configuración + evadir obstáculo)**

El robot debe visitar los puntos de la ruta (en orden) *al mismo tiempo* que evade los obstáculos que le estorban, la combinación de estos dos comportamientos da como resultado llegar a la configuración siguiente de la secuencia generada por el navegador.

El comportamiento de *alcanzar configuración* se activa cuando hay una nueva meta y utiliza las ecuaciones 3.13 y 3.14 para determinar el ángulo de giro y el desplazamiento del robot.

Por su parte el comportamiento de *evadir obstáculo* se activa cuando hay una meta, el robot no ha llegado a ella y está "apuntando" hacia ella. Este comportamiento le alimenta las lecturas de los sonares a una red neuronal de retropropagación, la cual responde con dos propuestas de giro: una a la izquierda y la otra a la derecha. Estos ángulos se utilizan para generar un par de posiciones parciales que sirven para "darle la vuelta" al obstáculo. Para determinar qué posición parcial es mejor, le consulta al planificador y finalmente define una sola posición parcial a alcanzar.

4.4 Especificación del entorno y del controlador

Una vez definidas las características iniciales del entorno así como las espectativas que se tienen sobre el robot, se puede definir con precisión la manera en que se utilizan los sensores y los actuadores. También se pueden definir las características que deben tener el entorno del robot y la arquitectura del controlador.

4.4.1 Sensores y actuadores

Aunque el robot cuenta con sonares, sensores infrarrojos, sensores de contacto y sensores de posición, para este trabajo se planteó utilizar solamente los sonares y el odómetro. Los sensores infrarrojos y de contacto podrían ayudar para detectar objetos y se podrían integrar al comportamiento de *evadir obstáculo*, pero para evaluar el uso de la red neuronal es suficiente por el momento con un tipo de sensor ya que es posible utilizar los otros sensores en una etapa posterior del proyecto que no cubre este trabajo.

Los ocho sonares frontales del robot son suficientes para saber de la existencia de algún obstáculo. El sonar es un sensor que presenta varios problemas [18], de los cuales el más importante es que puede entregar lecturas falsas. Por ejemplo, cuando el material que se le presenta es poco rugoso y no es perpendicular a la dirección de la emisión sonora, el haz se puede reflejar en una dirección tal que no regrese nunca al sensor, y se presenta una lectura nula aún habiendo un objeto frente al sonar. Otro de los problemas es que existe la posibilidad de que un haz sea reflejado por un objeto en otra dirección y posteriormente reflejado por otro objeto hacia el sensor, con lo cual también se tendría el problema de una lectura mayor a la distancia del obstáculo. Por otro lado el sonar no entrega lecturas uniformes en todo su rango (que es de alrededor de +/- 30°.

El odómetro se utiliza para determinar la posición y orientación del robot con respecto a una referencia inicial. Su principal problema es que basa sus lecturas en encodificadores, que miden el número de veces que giran las ruedas. Ésto puede llevar a lecturas imprecisas o erroneas, por la incertidumbre del radio de las ruedas o por los desniveles en el piso.

Los actuadores que tiene el robot, como ya se dijo, conforman un arreglo de tipo "synchro-drive". En la figura 4.3 se bosqueja la base del B14. En 4.3.a) el robot está orientado hacia la derecha y en 4.3.b) ha girado 45° sobre su eje en el sentido opuesto a las manecillas del reloj. Se ve cómo para cambiar de orientación, las tres ruedas cambian su orientación de manera sincrónica. Gracias a ésto el robot puede cambiar de orientación sin restricciones cinemáticas y puede alcanzar cualquier punto de su entorno con solo girar sobre su eje y desplazarse.

Las características del robot hacen innecesario alterar o añadir sensores o actuadores para cumplir con el objetivo de este trabajo.

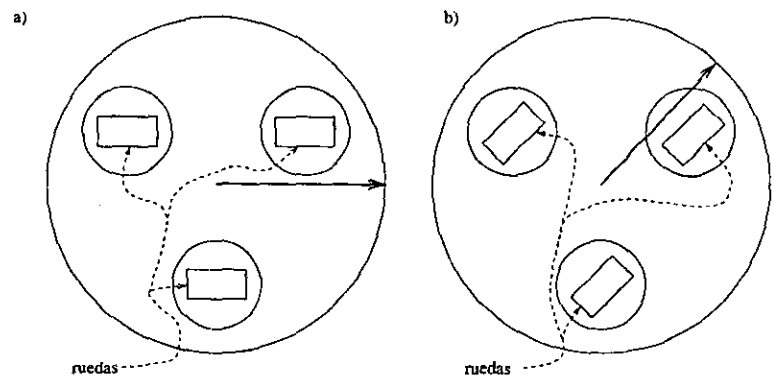


Figura 4.3: Ruedas en arreglo Synchro Drive

4.4.2 Entorno extendido

El entorno extendido es el resultado de modificar el entorno inicial para adecuarlo a las limitaciones del robot. Una de las motivaciones de este trabajo es contar con un robot que pueda operar en un entorno "natural", es decir, sin adaptarse para que el robot pueda realizar su misión. Es por esto que se decidió desde el principio no alterar el laboratorio y por lo tanto el entorno extendido es igual al entorno inicial.

4.4.3 Arquitectura del controlador

Para poder hacer un controlador híbrido, que aproveche el modelo tradicional y el de comportamientos, se realizó un controlador con una arquitectura distribuida en cinco módulos: el planificador, el navegador, el piloto, el controlador y la red neuronal.

En la figura 4.4 se observa la estructura de bloques de un agente robot. La idea es dar los primeros pasos para en un futuro llegar a esa estructura. El robot que se implanta en este trabajo es igual al de la figura, excepto porque le falta la realimentación de las líneas punteadas que van del simulador al bloque de modelo del mundo y de los sensores al modelo del mundo.

En la figura 4.5 se observa a detalle la distribución de los diferentes componentes del controlador de este trabajo. Se puede notar que el planificador y el navegador se integran en un solo bloque: el *planificador/navegador*. También se observa que el modelo del mundo no aparece, la razón es que está integrado también en el *planificador/navegador*, lo que se hace usando reglas de producción.

El *planificador* recibe del usuario instrucciones a través de una interfaz gráfica o directamente del teclado. Una vez que se le ordena llegar a una meta, busca una ruta óptima y libre de colisiones usando técnicas de planificación para que el robot cumpla con las instrucciones mediante el comportamiento de

planificar ruta y se la envía al *navegador*. También mantiene en el mapa las características del entorno del robot, de manera que puede informar al *piloto* o a cualquier programa que se lo pida, si una posición está dentro o fuera de un área inalcanzable. Cabe aclarar que aunque el *planificador* debería generar rutas globales y rutas locales, en éste trabajo sólo se llega a la generación de rutas locales.

El *navegador* divide la ruta en una secuencia de configuraciones. Si tiene comunicación con el *piloto* le manda una por una las configuraciones que éste debe alcanzar y espera a que éste le informe que ya alcanzó la configuración solicitada. Si no hay comunicación simula los movimientos del robot en la interfaz con el usuario.

El *piloto* cuenta con dos comportamientos: *alcanza configuración* y *evade obstáculo*. Éstos le permiten determinar los movimientos necesarios para llegar a una configuración evadiendo los obstáculos que encuentre, utilizando la información de los sonares y el odómetro del robot. Le manda al *controlador* las órdenes elegidas por el árbitro y le pide constantemente el estado de los sensores. Cuando alcanza la configuración solicitada se lo informa al *navegador* . Si el comportamiento *evade obstáculo* requiere saber si una posición es alcanzable para el robot se lo pregunta al *planificador*.

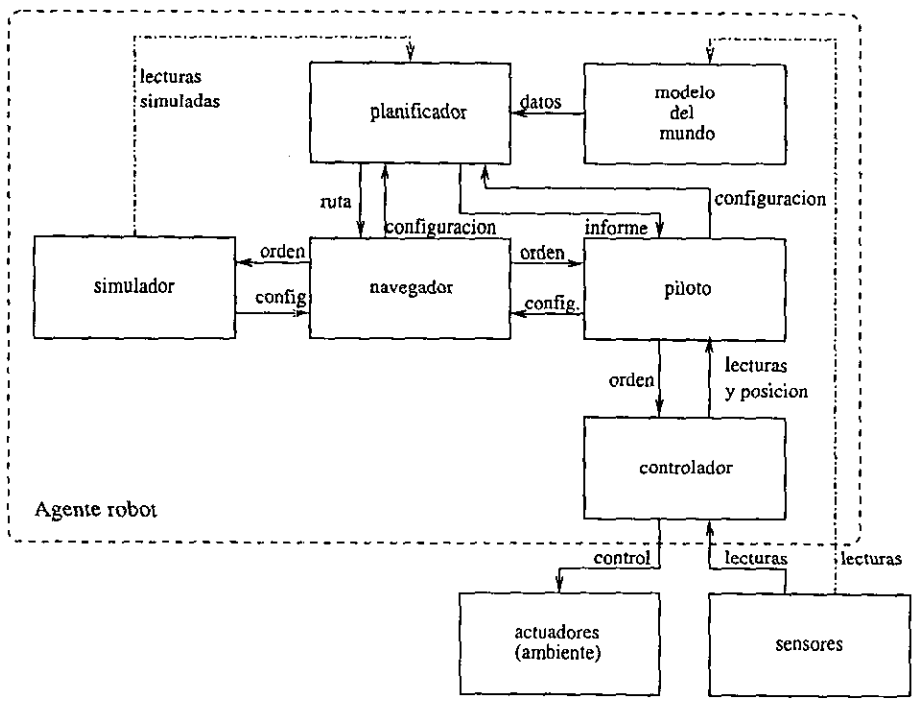


Figura 4.4: Estructura del controlador en bloques

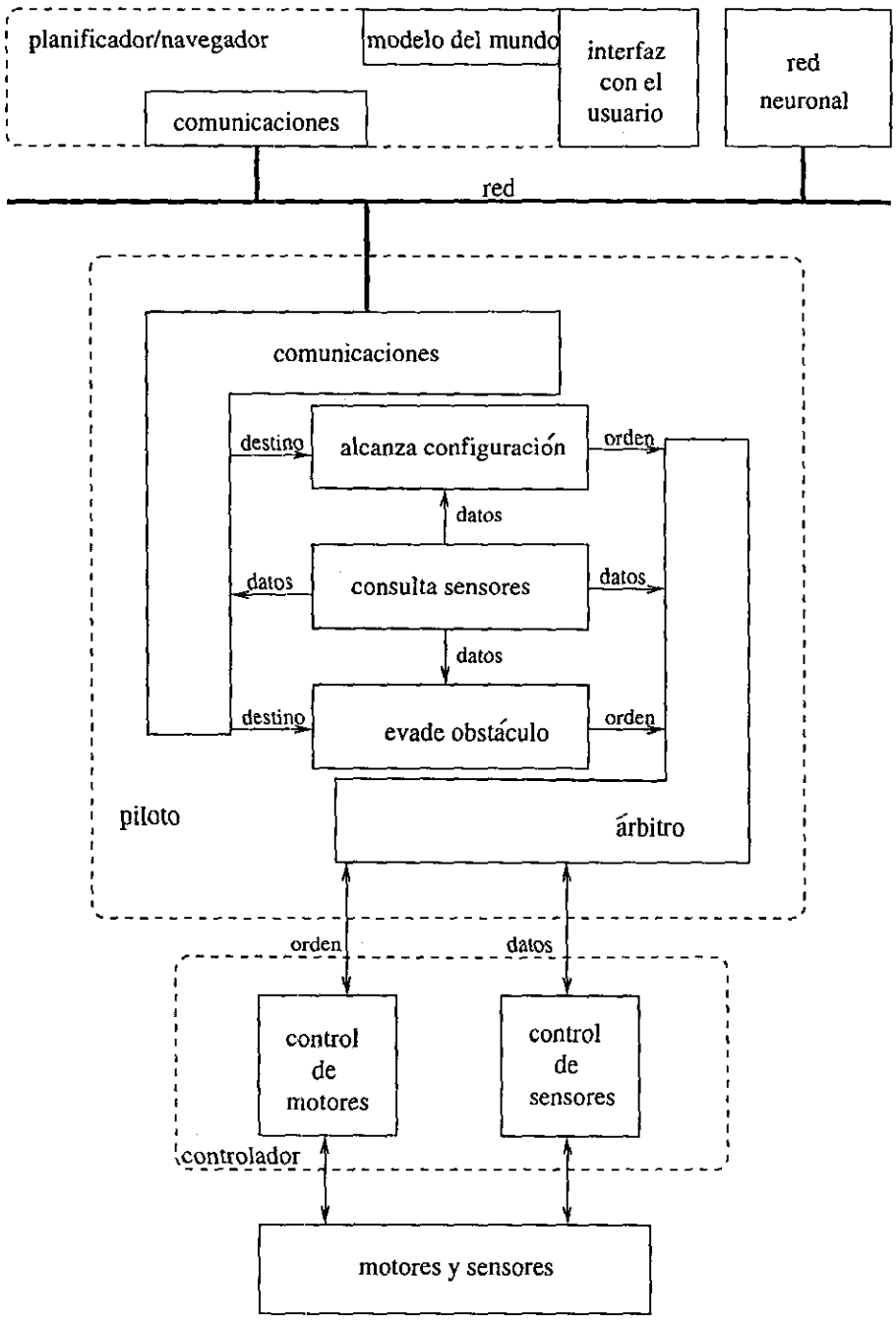


Figura 4.5: Arquitectura del controlador

El *controlador* es un módulo que ya se encuentra implementado en el robot B14. A través de su interfaz de programación de aplicaciones (API) se establecen la velocidad, aceleración y torque de giro y de desplazamiento y se envían las órdenes a la base, y también se pueden recuperar las lecturas de los sensores. La base del robot es controlada por el microcontrolador NEC 78310L que se encarga de mover los motores y controlarlos para lograr los giros y desplazamientos requeridos. Los dos microcontroladores MC68HC11 controlan a todos los sensores del robot y reportan, cuando se les requiere, los datos que éstos recogen.

La *red neuronal* procesa las lecturas de los sonares que le manda el piloto. Su salida está formada por dos datos: Cuanto debería girar el robot a su izquierda y cuánto debería girar a su derecha para dejar de tener frente a él un obstáculo. La red neuronal le dá servicio al comportamiento *evade obstáculo* del *piloto*.

4.4.4 Estrategia de entrenamiento

La cantidad de situaciones distintas a las que se puede enfrentar un robot móvil es inumerable. Los procesos que tratan con la información de sus sensores, en éste caso de los sonares, del odómetro e incluso del mapa que es contruido para el robot son los que tienen que lidiar con las incertidumbres

Las lecturas de los sonares se procesan utilizando una red neuronal, que es un elemento que de por sí se debe entrenar. Las lecturas del odómetro son procesadas por los dos comportamientos del piloto y por el momento no se plantea una estrategia de entrenamiento para éstos ni tampoco para minimizar los errores del mapa.

El único módulo que se sometió a entrenamiento fué la red neuronal, que ofrece alternativas de giro al comportamiento *evadir obstáculo*. El hecho de no entrenar todos los comportamientos limita al robot en varios sentidos, ya que no se define un mecanismo que permita mejorar el desempeño del robot en forma global. Pero dado que se está utilizando una arquitectura de control mixta y no el enfoque de comportamientos puro, se cuenta con un módulo capaz de planificar acciones de alto nivel reduciendo un poco esta desventaja.

La red neuronal recibe las lecturas de los ocho sonares frontales del robot, y genera dos propuestas de giro, una a la izquierda y el otro a la derecha. Se utilizó una red neuronal de retropropagación, por lo que el entrenamiento se tuvo que realizar *off-line*. Para realizarlo se tomaron lecturas de los sonares ubicando al robot en distintas situaciones tipo y se indicaron las respuestas que se esperaban de él.

4.5 Elección de la plataforma de desarrollo

Las características del robot definen la plataforma de desarrollo de Hardware y de Software casi en su totalidad. De hecho algunos de los módulos del controlador están practicamente definidos de antemano.

Debido a que el robot cuenta con una computadora Pentium con Sistema Operativo Linux, la plataforma de desarrollo de Software y Hardware de todos los módulos debe ser ésta o alguna compatible.

Para realizar el *planificador* y el *navegador* se decidió utilizar un sistema experto, ya que las características de éstos son muy adecuadas para lidiar con la naturaleza incierta y cambiante del entorno en que trabajan los robots móviles. Además es posible introducir conocimientos sobre la planificación y navegación sin requerir de un fuerte control sobre la secuencia de ejecución de las reglas, ésto facilita el crecimiento modular del sistema.

Para implantar el sistema experto se decidió utilizar CLIPS, la herramienta de desarrollo de Sistemas Expertos descrita en el capítulo 2, ya que sus características son ideales para el desarrollo. Gracias a que la comunicación entre los módulos se realiza a través de la red, se puede correr el Sistema Experto en cualquier computadora que tenga conexión en red con el robot. De hecho en la etapa de implantación se utilizó una Workstation Digital con procesador Alpha, y posteriormente se migró el programa al robot.

Para implantar al *piloto* se utilizó el API (Aplications Programming Interface) del B14 [19], que permite instruir al controlador mediante la llamada a funciones incluidas. Estas funciones permiten hacer que el robot se desplaze o gire, controlar su velocidad, aceleración y torque y obtener las lecturas de los sensores. También tiene un "scheduler" que es una tabla cuyos campos son funciones y el momento en que se deben ejecutar. Ésto permite que el programador defina la ejecución de funciones de manera periódica o como respuesta a señales de los sensores o eventos sin necesidad de definir ciclos estrictos de ejecución.

El "scheduler" facilita definir los módulos generadores de comportamiento así como el árbitro (necesarios para programar un robot usando el enfoque de comportamientos). La comunicación entre los comportamientos y el árbitro se realizó mediante el uso de variables globales en el programa del piloto.

El *controlador* es el que ya está definido en el robot y simplemente se utilizó.

La *red neuronal* se puede implementar de muchas formas y en diversas plataformas. Dado que el interés en éste trabajo es probar el funcionamiento de las mismas en un robot y no desarrollar redes neuronales, se decidió utilizar software que está disponible en Internet: ASPIRIN/Migraines [20]. Éste programa de distribución gratuita sirve para simular redes neuronales. Incluye una red neuronal de retropropagación, aunque permite definir la estructura de otros tipos de redes neuronales. Su principal ventaja es que se puede definir una red neuronal con sólo describir en un archivo las características de la misma, a saber: número de neuronas por capa, función de activación de las neuronas y número de capas.

Capítulo 5

Diseño e implantación de los módulos para el control del robot

En éste capítulo se cubre el quinto punto del método descrito en el capítulo anterior: Diseño e implantación de los módulos para el control del robot.

5.1 Planificador

El planificador es el agente que tiene las siguientes tareas:

- Mantiene una representación a priori del entorno del robot en la que se incluyen los obstáculos, los límites del área en donde se mueve el robot y la configuración del mismo.
- Genera las áreas prohibidas para el robot y produce un mapa de carreteras.
- A partir del mapa de carreteras planifica una ruta para llegar a la configuración meta.
- Interactuar con el usuario para que éste le indique la nueva meta del robot.
- Es capaz de responder a solicitudes de información sobre alguna posición indicando si ésta es alcanzable o no y que tan lejos se encuentra del objetivo.

Para cumplir con sus tareas se crearon varios grupos de reglas en el shell para sistemas expertos CLIPS [1], que a continuación se detallan.

[1] en las siguientes secciones se presentan fragmentos de programas codificados en CLIPS, para más información sobre éste lenguaje se puede consultar el apéndice A

5.1.1 Representación del entorno del robot

Para representar el entorno del robot se utiliza el concepto del *espacio de configuraciones* del cual se habló en el capítulo 3. El entorno del robot está poblado de objetos, que para él son obstáculos que limitan su capacidad de movimiento. Ya que el robot sólo se puede mover en un plano, se utilizan polígonos para representar los obstáculos. Estos polígonos se representan a través de una lista ordenada de sus vértices, encabezada por el nombre del polígono y seguida por las coordenadas (x, y) de todos y cada uno de los vértices del mismo.

Cada objeto situado en el entorno del robot se identifica mediante un hecho en la memoria de trabajo de CLIPS que cumple con el siguiente patrón:

```
(polygonal_object ?nombre ?x1 ?y1 ?x2 ?y2 . . . ?xn ?yn)
```

Donde `?nombre` es el nombre con el que se identificará al polígono, y las parejas `?xi ?yi` son los vértices del mismo ordenados en el sentido de las manecillas del reloj. De aquí en adelante, todos los símbolos precedidos por un signo de interrogación(?) son variables, siguiendo la notación propia de CLIPS.

También es necesario definir los límites del área de trabajo del robot, ésto se hace con el patrón:

```
(limit_area ?x1 ?y1 ?x2 ?y2 . . . ?xn ?yn)
```

En este caso también las parejas `?xi ?yi` son los vértices del área y están ordenadas en el sentido de las manecillas del reloj. No es necesario un nombre para identificar esta zona pues sólo hay una.

Para representar los objetos del Laboratorio de Interfaces Inteligentes tal y como se observa en la figura 4.2 del capítulo 4 se utiliza un archivo que es cargado posteriormente en el sistema experto con los siguientes hechos:

```
(limit_area 0.0 0.0 0.0 7.42 7.11 7.42 7.11 0.0)

(polygonal_object mesa_1 0.0 2.2 0.0 2.9 3.0 2.9 3.0 2.2)
(polygonal_object mesa_2 0.0 2.9 0.0 5.9 0.7 5.9 0.7 2.9)
(polygonal_object mesa_3 0.5 7.4 3.5 7.4 3.5 6.6 0.5 6.6)
(polygonal_object mesa_4 4.1 7.4 7.1 7.4 7.1 6.6 4.1 6.6)
(polygonal_object mesa_5 6.3 5.9 7.1 5.9 7.1 2.9 6.3 2.9)
(polygonal_object mesa_6 7.1 2.9 7.1 2.2 4.1 2.2 4.1 2.9)
(polygonal_object caja   3.0 4.0 2.5 4.0 2.5 4.6 3.0 4.6)
```

Si se quiere incluir más elementos basta con agregar las líneas correspondientes. Si por otra parte, se desea alimentar un mapa de un lugar distinto, se puede usar éste como ejemplo para crearlo.

5.1.2 Generación de áreas prohibidas y del mapa de carreteras

Las reglas de generación de áreas prohibidas y del mapa de carreteras tienen las siguientes funciones:

- Leer el mapa del entorno del robot.
- Construir los *Obstáculos-C*.
- Definir los nodos de los *Obstáculos-C* que el robot puede alcanzar.
- Realizar el mapa de carreteras (roadmap).

El mapa del entorno del robot se almacena en un archivo, como ya se mencionó y se carga en el sistema experto como un conjunto de hechos.

Construcción de los *Obstáculos-C*

Las áreas prohibidas para el robot son los *Obstáculos-C* que se generan ampliando los polígonos en una distancia mayor o igual al radio del robot, como se puede ver en la figura 5.1. La representación de cada *Obstáculo-C* se obtiene trazando rectas paralelas a la frontera del obstáculo y localizando las intersecciones entre todas las rectas generadas. Aunque éste método contempla como espacio prohibido una pequeña área en las esquinas de algunos *Obstáculos-C* que en realidad es alcanzable, la representación es simple.

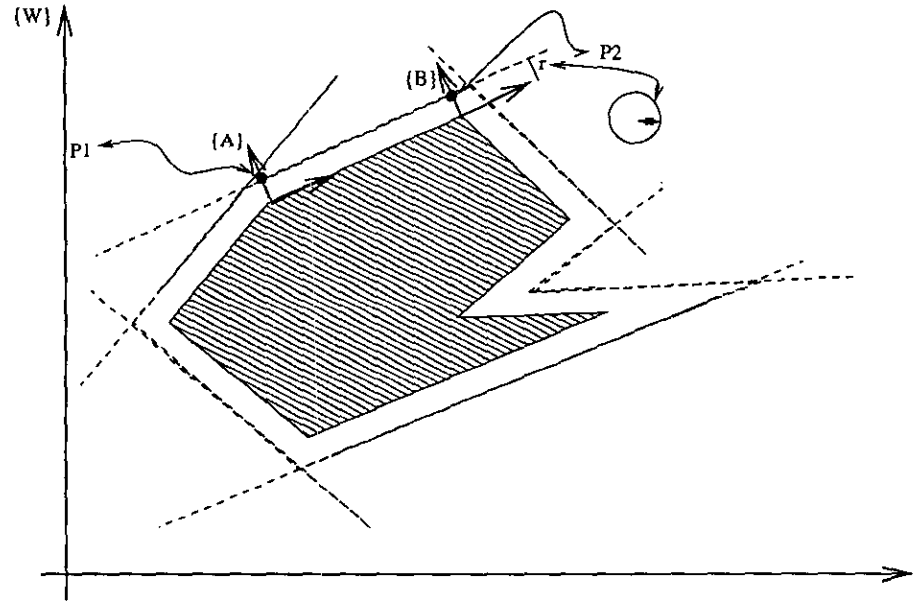


Figura 5.1: Obstáculos-C

Para hacer la expansión se necesita encontrar una recta paralela a cada una de las que forman el polígono. Se utiliza el concepto de transformación delineado en la sección 3.6.1 del capítulo 3, para determinar la posición de un punto sobre el eje y de un sistema de referencia cuyo origen está en el vértice i del polígono

y su orientación es la misma que la del vector que se puede trazar del vertice i al vértice $i+1$. La distancia a la que se encuentra el punto es igual a r. Dado ésto se tiene la siguiente ecuación:

$$^{W}P_i = ^{W}P_{org(N)} + ^{W}_{N}R\ ^{N}P_i$$

donde W es el sistema de referencia general y N es el sistema de referencia cuyo origen está en el punto a partir del cual se quiere obtener el punto i.

Desarrollando esta ecuación se obtiene:

$$^{W}P_i = \begin{bmatrix} x_i - r\sin\theta \\ y_i - r\cos\theta \end{bmatrix} \tag{5.1}$$

Una vez que se obtiene la nueva pareja de puntos se encuentra la siguiente y se localiza la intersección entre las dos rectas generadas. El punto encontrado es uno de los vértices del *Obstáculo-C*, los demás vértices se obtienen usando el mismo proceso.

El entorno del robot está limitado por otro polígono cuyo perímetro coincide con las paredes del lugar. Para conocer el área donde se puede mover el robot es necesario reducir este polígono de manera similar a como los obstáculos se agrandaron usando la ecuación 5.1 también, pero con una r negativa.

A continuación se muestra la función `alter_polygon` codificada en CLIPS, que agranda o reduce un polígono, dependiendo del signo del parámetro `dist`, si es positivo se agranda el polígono, si es negativo, se empequeñece. La magnitud de `dist` es la distancia que se quiere modificar el polígono. El otro parámetro que recibe es la lista de los vértices del polígono, ordenada en el sentido de las manecillas del reloj, en forma de pares (x, y). La ecuación 5.1 se aplica dentro de la función `paralel` para encontrar la recta paralela a la definida entre cada par de vértices. La función `intersection_point` determina los vértices del nuevo polígono.

```
(deffunction alter_polygon (?dist $?vertex)
 (bind ?size (length ?vertex))
 (bind ?i 1)
 (bind ?new_vertex (create$))
 (bind ?xin (bind ?x1 (nth$ ?i ?vertex)));x 1er vértice
 (bind ?yin (bind ?y1 (nth$ (+ ?i 1) ?vertex)));y 1er vértice
 (bind ?x2 (nth$ (+ ?i 2) ?vertex));x 2o vértice
 (bind ?y2 (nth$ (+ ?i 3) ?vertex));y 2o vértice
 ;obtiene paralela al segmento entre (x1,y1) y (x2,y2)
 (bind ?straight_1 (bind ?straight_A
                   (paralel ?dist ?x1 ?y1 ?x2 ?y2)))
 (bind ?i (+ ?i 4)) ;ciclo para obtener los demás vértices
 (while (< ?i (+ ?size 2))
  (bind ?x1 ?x2)
  (bind ?y1 ?y2)
  (if (< ?i ?size) then
```

```
   (bind ?x2 (nth$ ?i ?vertex))
   (bind ?y2 (nth$ (+ ?i 1) ?vertex))
  else            ; último vértice
   (bind ?x2 ?xin)
   (bind ?y2 ?yin))
  ;obtiene paralela
  (bind ?straight_B (paralel ?dist ?x1 ?y1 ?x2 ?y2))
  ;nuevo vértice
  (bind ?inter (intersection_point
        (expand$ ?straight_A) (expand$ ?straight_B)))
  ;agrega el nuevo vértice a la lista
  (bind ?new_vertex (create$
                     ?new_vertex (rest$ ?inter)))
  (bind ?i (+ ?i 2))
  (bind ?straight_A ?straight_B))
 (bind ?inter (intersection_point (expand$ ?straight_A)
                                  (expand$ ?straight_1)))
 ;regresa los vértices del nuevo poligono
 (bind ?new_vertex (create$
       ?new_vertex (rest$ ?inter)))
)
```

Una vez que se definió esta función se realizó una regla que se activa cada que hay una nueva área definida, que no se ha construido un *obstáculo-C* a partir de ella y que existe un robot y un factor de seguridad para poder llamar a la función `alter_polygon`. Una regla similar sirve para definir el espacio en que se le permite al robot moverse, pero a partir de los límites definidos con anterioridad. Este par de reglas generan los *Obstáculos-C* que en el programa se definieron como áreas prohibidas.

Definición de los nodos alcanzables

Dadas las condiciones anteriores y que el robot no presenta restricciones cinemáticas, el espacio de configuraciones que se obtiene es de dos dimensiones, ya que para alcanzar cualquier configuración basta conocer las coordenadas x, y de ésta. La orientación del robot no importa para determinar si una posición es alcanzable o no. Por todas estas razones se eligió utilizar el método de *mapa de caminos* para realizar la planificación de movimientos. En este método es necesario realizar un grafo de visibilidad cuyos nodos son los vértices de todos los polígonos que están fuera de los *obstáculos-C*, además de las posiciones inicial y meta (figura 5.2.

Una vez definidos los *Obstáculos-C*, se realiza una lista de todos los vértices de todos los obstáculos y se busca cuáles están fuera de todos y cada uno de ellos.

Para ésto se definió una regla que hace la lista de todos los nodos. Esta regla se activa una vez por cada área prohibida y modifica un hecho que almacena

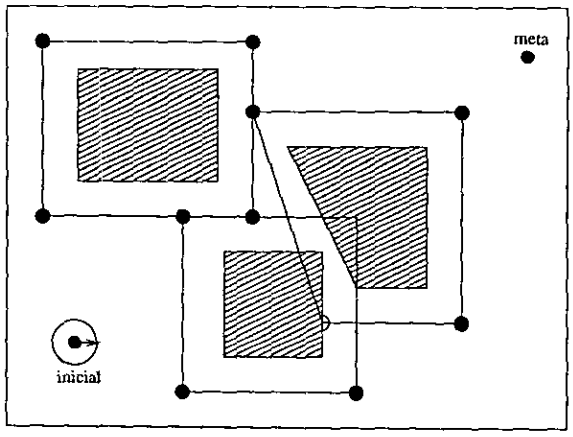


Figura 5.2: Nodos del grafo de visibilidad realizados por el planificador

los pares de puntos que no se repiten. Otra regla verifica los puntos que están dentro de los límites permitidos.

También se definió otra regla que se activa una vez por cada área prohibida, y que usa la lista de nodos que están dentro de los límites para definir los puntos que están fuera de cada una de las áreas prohibidas. En el lado derecho de esta regla se buscan los puntos que están fuera del polígono. A continuación se puede ver su definición:

```
(defrule make_list_of_points_out_of_forbidden_areas
 (forbidden_area ?name $?vertex); área prohibida
 (points_in_allowed_area $?point); puntos a verificar
=>
 (bind ?i 1)
 (bind ?size (length ?point))
 (bind ?newlist (create$))
 (while (< ?i ?size); ciclo de verificación
  (bind ?x (nth$ ?i ?point))
  (bind ?y (nth$ (+ ?i 1) ?point))
  (bind ?i (+ ?i 2))
  (bind ?outside
    (evaluate_if_inside ?x ?y ?vertex)); evalúa si el punto está
                                       ; dentro del área
  (if (>= ?outside (- 0 .0001)) then ; está fuera
   (bind ?newlist (create$ ?newlist ?x ?y))))
 (assert (points_out_of ?name ?newlist))
)
```

La función `evaluate_if_inside` recibe las coordenadas (x, y) del punto y la lista de vértices que definen al polígono. Regresa cero cuando el par (x, y) está

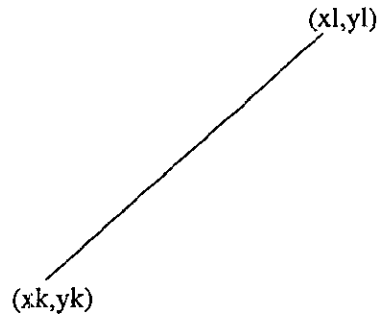


Figura 5.3: Recta definida por un par de puntos: (x_k, y_k) y (x_l, y_l)

sobre la frontera del polígono definido por los vértices en la lista `vertex`. Si el punto está dentro del polígono regresa un número menor que cero y si está fuera del mismo, regresa un número mayor que cero, en ambos casos la magnitud del número es la distancia a la que se encuentra el punto del lado del polígono más cercano.

Para determinar si el punto está dentro, fuera o sobre la frontera del polígono se utilizan las ecuaciones paramétricas que definen cada una de las rectas que conforman el polígono, como se muestra en la figura 5.3. Las ecuaciones se definen de la siguiente manera:

$$t = \frac{x - x_k}{x_l - x_k} = \frac{y - y_k}{y_l - y_k} \tag{5.2}$$

donde (x_k, y_k) son las coordenadas del primer punto del segmento y (x_l, y_l) son las coordenadas del último punto, t por su parte es el parámetro.

Resolviendo la segunda igualdad queda:

$$y(x_l - x_k) + x(y_k - y_l) + (x_k y_l - y_k x_l) = 0 \tag{5.3}$$

Si se sustituyen en esta ecuación las coordenadas (x, y) del punto que se quiere probar si está a su "izquierda" (lo cual sería equivalente a que está fuera del polígono, al menos con respecto al lado definido por el segmento de recta), el resultado se confirma en cero cuando el punto está sobre el segmento, mayor a cero cuando está a su "izquierda" y menor a cero cuando a su "derecha" (o dentro del polígono con respecto al lado que se está considerando). Ya que la lista de vértices está ordenada en el sentido de las manecillas del reloj, basta con hacer pruebas para cada pareja de puntos. Si alguna prueba reporta que el punto está fuera del polígono, entonces se puede afirmar que en efecto el punto se encuentra fuera. Sólo si *todas* las pruebas reportan que el punto está dentro del polígono se puede afirmar que lo está.

```
(deffunction evaluate_if_inside (?x ?y $?vertex)
 (bind ?i 1)
```

```
(bind ?greater -100); debería ser -infinito
(bind ?number_of_vertex (length ?vertex))
(bind ?vertex (create$ ?vertex (subseq$ ?vertex 1 2)))
(while (< ?i ?number_of_vertex); cliclo de evaluación
 (bind ?xk (nth$ ?i ?vertex))
 (bind ?yk (nth$ (+ ?i 1) ?vertex))
 (bind ?xl (nth$ (+ ?i 2) ?vertex))
 (bind ?yl (nth$ (+ ?i 3) ?vertex))
 (bind ?res (+ (+ (* ?y (- ?xl ?xk)) (* ?x (- ?yk ?yl)))
                          (- (* ?xk ?yl) (* ?yk ?xl))))
 (if (> ?res 0) then; fuera del polígono
  (bind ?greater ?res)
  (bind ?i ?number_of_vertex)
 else
  (if (> ?res ?greater) then; el resultado es la dist. mayor
   (bind ?greater ?res)))
 (bind ?i (+ ?i 2)))
(bind ?result ?greater)
)
```

Una vez que se han evaluado todas las áreas prohibidas definidas, se hace una lista que incluye la lista de todos los nodos que están fuera de todas ellas haciendo la intersección entre todas las listas generadas. El resultado es un hecho `(roadmap_node ?nombre ?x ?y)` por cada uno de los puntos de la intersección.

Construcción del mapa de carreteras

El mapa de carreteras es un grafo de visibilidad cuyos nodos son los que se definieron en el punto anterior. Ahora se debe evaluar si hay o no conexión entre un par de nodos. En primera instancia se evalúa si la recta definida entre cada par de nodos cruza por un área prohibida.

Si los nodos pertenecen al área prohibida contra la que se hace la evaluación y no se encuentran en posiciones consecutivas en la lista de puntos que la definen, entonces se afirma que el área prohibida está entre ellos. Si no pertenecen al área prohibida se prueba si el segmento de recta entre los nodos se intersecta con alguno de los segmentos de recta que definen al polígono. Si hay intersección, se afirma también que el área prohibida está entre ellos. La regla que se muestra a continuación realiza lo descrito:

```
(defrule identify_forbidden_areas_crossing_routes
 (roadmap_node ?name1 ?x1 ?y1); nodo 1
 (roadmap_node ?name2 ?x2 ?y2); nodo 2
 (test (neq ?name1 ?name2)); verifica que el nodo 1 y el 2
                           ; no son el mismo
 (forbidden_area ?area_name $?point); área a evaluar
 (not (between ?name1 ?name2 ?area_name))
 (not (between ?name2 ?name1 ?area_name))
```

```
=>
 (bind ?size (length ?point))
 (bind ?point (create$ ?point (subseq$ ?point 1 2)))
 (bind ?position_x_node_1 0)
 (bind ?position_x_node_2 0)
 (bind ?i 1)
 ;verifica de puntos consecutivos que pertenecen al área
 (while (<= ?i ?size )
  (bind ?x_temp (nth$ ?i ?point))
  (bind ?y_temp (nth$ (+ ?i 1) ?point))
  (if (and (= ?x_temp ?x1) (= ?y_temp ?y1)) then
   (bind ?position_x_node_1 ?i))
  (if (and (= ?x_temp ?x2) (= ?y_temp ?y2)) then
   (bind ?position_x_node_2 ?i))
  (bind ?i (+ ?i 2)))
 (if (and (= ?position_x_node_1 0)
          (= ?position_x_node_2 0)) then
  (bind ?verify 1)
  (bind ?except 0)
 else
  (if (or (= ?position_x_node_1 0)
          (= ?position_x_node_2 0)) then
   (bind ?verify 1)
   (bind ?except (+ ?position_x_node_1
                    ?position_x_node_2))
  else
   (bind ?difference (abs (- ?position_x_node_1
                             ?position_x_node_2)))
   (if (or (= ?difference 2)
           (= ?difference (- ?size 2))) then
    (bind ?verify 0)
   else
    (bind ?verify 0) ;área entre los nodos
    (assert (between ?name1 ?name2 ?area_name)))))
 ;si al menos un nodo no pertenece al área
 (if (= ?verify 1) then
  (bind ?i 1)
  (while (< ?i ?size)
   (if (not (or (= ?i ?except) (= (+ ?i 2) ?except)
            (and (= ?except 1) (= ?i (- ?size 1))))) then
    (bind ?xa (nth$ ?i ?point))
    (bind ?ya (nth$ (+ ?i 1) ?point))
    (bind ?xb (nth$ (+ ?i 2) ?point))
    (bind ?yb (nth$ (+ ?i 3) ?point))
    ;verifica intersección entre rectas
    (bind ?intersect (nth$ 1 (intersection_point
```

```
        ?x1 ?y1 ?x2 ?y2 ?xa ?ya ?xb ?yb)))
    (if (> ?intersect 0) then; área entre los nodos
     (assert (between ?name1 ?name2 ?area_name))
     (bind ?i ?size)))
   (bind ?i (+ ?i 2))))
)
```

Aquí se puede ver como también se utiliza la función `intersection_point` de la que ya se había hablado. Esta función detecta si hay intersección entre los dos segmentos de recta definidos por `?x1 ?y1 ?x2 ?y2 ?xa ?ya ?xb ?yb` donde los primeros dos pares de puntos se refieren al primer segmento y los otros dos al segundo segmento. Esta función regresa un cero cuando los segmentos coinciden al menos en un par de puntos, un uno cuando hay intersección entre los segmentos *dentro* del intervalo definido por ambos. Si no se cumple alguna de las dos condiciones anteriores regresa -1.

Una vez que se han definido las áreas que se cruzan entre un par de nodos se definen conexiones entre todos aquellos que no cumplen con ésta restricción con la siguiente regla:

```
(defrule define_route_between_nodes
 (declare (salience -1))
 (roadmap_node ?name1 ?x1 ?y1); nodo 1
 (roadmap_node ?name2 ?x2 ?y2); nodo 2
 (test (neq ?name1 ?name2)); no son el mismo nodo
 ;no hay polígonos entre ellos
 (not (between ?name1 ?name2 ?polygon_a))
 (not (between ?name2 ?name1 ?polygon_b))
 ;no se ha definido conexión entre ellos
 (not (roadmap_pathway ?name1 ?name2 ?distance_a))
 (not (roadmap_pathway ?name2 ?name1 ?distance_b))
=>
 (bind ?distance (distance_between ?x1 ?y1 ?x2 ?y2))
 ;se define la conexión
 (assert (roadmap_pathway ?name1 ?name2 ?distance))
)
```

Se puede ver que por cada conexión entre un par de nodos se define un hecho de la siguiente manera:

```
(roadmap_pathway ?nodo1 ?nodo2 ?distancia)
```

En el que `?nodo1` y `nodo2` son los nombres de los nodos que tienen conexiones, la `?distancia` es la distancia en linea recta que hay entre ellos.

5.1.3 Búsqueda de la mejor ruta

A partir del grafo de visibilidad el *planificador* realiza una búsqueda de la ruta a seguir utilizando un método similar al $A*$, que genera la mejor ruta para

alcanzar la configuración meta partiendo de la configuración inicial. El método empleado no garantiza encontrar la mejor ruta siempre, pero sí encuentra una de las mejores rutas con un costo de cómputo menor a $A*$.

Para encontrar el mejor camino se definió un conjunto de reglas que realiza lo siguiente:

- Evalúan si la configuración meta es alcanzable.
- Introducen los nodos origen y meta como nodos.
- Construyen un árbol de búsqueda con los nodos definidos por las reglas de creación de nodos.

Evaluación de la configuración meta

Una vez que se tiene una configuración meta, se evalúa si ésta es alcanzable. Se definió una regla que se activa una vez por cada una de las áreas prohibidas y para el área permitida, esta regla llama a la función `evaluate_if_inside` ya descrita y en cuanto detecta que la meta se encuentra dentro de algún área prohibida o fuera del área permitida, cancela la búsqueda de un camino y permite al usuario introducir una nueva configuración meta.

Introducción de los nodos origen y meta como nodos

Para poder iniciar la búsqueda de la mejor ruta es necesario introducir los nodos origen y meta al mapa de carreteras. Debido al funcionamiento de los sistemas expertos basados en reglas, basta con introducir el hecho que afirme que hay un nodo en las coordenadas (x, y) del origen y de la meta y las reglas descritas en la sección anterior hacen el resto. La siguiente regla cumple con este proposito:

```
(defrule set_nodes_target_and_source
 ;hay orden de ir a otro lado
 (order ?name go_to ?xt ?yt ?orientation_o)
 ;configuración del robot
 (robot ?name ?radius ?xs ?ys ?orientation_s)
 (not (path $?resolved))
=>
 ;introduce el origen como nodo
 (assert (roadmap_node target ?xt ?yt))
 ;introduce al destino como nodo
 (assert (roadmap_node source ?xs ?ys))
)
```

Construcción del árbol de búsqueda

El árbol está formado por nodos que contienen la siguiente información:

- Nombre del nodo.

- Nombre del nodo padre.
- Distancia acumulada desde el nodo raiz.
- Nivel en el que está el nodo.

Toda esta información se introduce en hechos como el que sigue:

```
(son_father_distance_level ?nombre ?padre ?distancia ?nivel)
```

y la búsqueda dá inicio una vez que está definido el hecho:

```
(find_a_path ?origen ?destino)
```

que fué introducido por la regla que recibe la orden de parte del usuario. `?origen` y `?destino` deben ser nodos definidos con las reglas de construcción de nodos de las que ya se habló. El nodo raiz tiene como nombre el del nodo origen, su padre es none, la distancia es cero y su nivel también es cero. Un hecho controla el nodo que se debe expandir:

```
(expand ?nodo ?nivel ?distancia)
```

Las variables `?nodo`, `?nivel` y `?distancia` sirven para identificar adecuadamente al nodo que se debe expandir y evitar ciclos en el árbol. La regla que sirve para expandir los nodos en el árbol es:

```
(defrule expand_node
 (declare (salience 1))
 ;lista de hojas no expandidas
 ?leaf_fact <- (leaf_distance_level $?list)
 (expand ?son ?level ?distance_to_father); nodo a expandir
 (son_father_distance_level ?son ?father
?distance_to_father ?level); datos del nodo
 ; hay ruta a otro nodo
 (or (roadmap_pathway ?son ?node ?distance)
     (roadmap_pathway ?node ?son ?distance))
 (not (visited ?node)); no se ha visitado el otro nodo
 (not (expanded ?node)); ni expandido (evita ciclos)
=>
 (bind ?heuristic_outcome (heuristic
       ?distance_to_father ?distance)); encuentra distancia
 ;calcula el nivel de la nueva hoja
 (bind ?node_level (+ ?level 1))
 ; introduce la hoja al árbol
 (assert (son_father_distance_level
   ?node ?son ?heuristic_outcome ?node_level))
 (assert (visited ?node))
 ;introduce la hoja a la lista de hojas
 (assert (leaf_distance_level $?list ?node
```

```
        ?heuristic_outcome ?node_level))
 (retract ?leaf_fact)
)
```

Esta regla se activa una vez por cada conexión que tiene el nodo a expandir con otros nodos que *no* han sido expandidos previamente ni han sido visitados. En el lado izquierdo de esta regla se usa el hecho `(leaf_distance_level ?list)`, éste contiene una lista de nodos que no se han expandido, por lo que en el lado derecho se agrega el nuevo nodo a la lista. En ésta lista se introduce el nombre del nodo, el nombre de su padre, la distancia y el nivel. La distancia heurística que se introduce no es la distancia acumulada sino una subestimación de la distancia total que el robot tiene que recorrer si sigue ese camino.

Cada que se expande un nodo se ordena `(leaf_distance_level ?list)` para garantizar que al principio se encuentra la hoja con la menor distancia acumulada. La siguiente regla llama a una función de ordenamiento y cumple con este propósito:

```
(defrule order_the_leafs
 ?leaf_fact <- (leaf_distance_level $?elements); lista de hojas
 (not (leaf_distance_ordered $?in_order)); no hay lista ordenada
 (not (visited ?node))
=>
 (retract ?leaf_fact)
 (bind ?leaf_ordered (order_leafs $?elements)); ordena la lista
 ;introduce la lista ordenada
 (assert (leaf_distance_ordered $?leaf_ordered))
)
```

Para encontrar el nodo que se debe expandir sólo se extrae de la lista ordenada de hojas el primer nodo y se introduce en la memoria de trabajo el hecho que ordena que se debe expandir. La siguiente regla realiza estos pasos:

```
(defrule find_the_most_feasible_node
 ?ordered <- (leaf_distance_ordered ?first
              ?distance ?level $?others); lista ordenada
=>
 (retract ?ordered )
 ;ordena expandir el 1er nodo
 (assert (expand ?first ?level ?distance))
 ;extrae el 1er nodo de la lista
 (assert (leaf_distance_level $?others))
)
```

Cada vez que se ordena expandir un nodo se busca si el nodo meta es su hijo. Si ésto sucede se suspende la búsqueda y se reconstruye el camino en el árbol para encontrar la ruta. Esto se logra usando la siguiente regla:

```
(defrule check_for_the_target
 (declare (salience 2))
 ;nodo a expandir
 ?expanding <- (expand ?node ?level ?distance_to_father)
 ?last <- (last_node ?probe); nodo meta
 ;hay camino al nodo meta
 (or (roadmap_pathway ?node ?probe ?distance)
     (roadmap_pathway ?probe ?node ?distance))
 ?leaf <- (leaf_distance_level $?members)
 (son_father_distance_level ?node ?father
          ?distance_to_father ?level)
 (expanded ?father)
=>
 (retract ?expanding ?last ?leaf); suspende la búsqueda
 ;inicia la reconstrucción de la ruta encontrada
 (assert (level_path (- ?level 1) ?father ?node ?probe))
)
```

A través del hecho `(level_path (- ?level 1) ?father ?node ?probe)` se le indica al sistema experto que debe encontrar el padre del nodo que es meta. La siguiente regla es la que reconstruye la ruta:

```
(defrule make_path
 ?path <- (level_path ?level ?son $?others);ruta a reconstruir
 ;padre del primero
 (son_father_distance_level ?son ?father ?distance ?level )
 (expanded ?father); el padre ya se expandió
=>
 (retract ?path)
 (bind ?fathers_level (- ?level 1)); reduce el nivel de la ruta
 ;aumenta el padre del primero a la ruta
 (assert (level_path ?fathers_level ?father ?son $?others))
)
```

Esta regla usa la información de los nodos para encontrar cual es su padre, la ruta es almacenada en el hecho `(level_path ?fathers_level ?father ?son $?others);$`. El proceso de reconstrucción del camino se detiene cuando se llega al nodo raiz. En ese momento se declara que la búsqueda ha terminado y se informa la ruta al navegador.

5.1.4 Interfaz con el usuario

Se plantearon dos formas de interacción con el usuario: la primera consiste en que el usuario alimente la nueva meta mediante el teclado; y la segunda le permite al usuario utilizar una interfaz gráfica en la que se muestran a escala todos los objetos del entorno del robot y le puede indicar al robot la configuración meta con un dispositivo señalador.

Para interactuar con el usuario se definió un conjunto de reglas con las siguientes funciones:

- Pedir al usuario el nombre del archivo donde están definidos los obstáculos y el área de trabajo y cargarlo en la memoria de trabajo.
- Permitir al usuario definir la configuración inicial del robot y su radio, así como una distacia de tolerancia que se utiliza en la construcción de los *Obstáculos-C*.
- Permitir al usuario definir si quiere ver en movimiento al robot o sólo quiere hacer una simulación.
- En caso de que el usuario haya decidido ver en movimiento al robot, se establece la comunicación con el robot
- Permitir al usuario establecer que tipo de interfaz quiere utilizar: de texto o gráfica.
- Si el usuario eligió usar la interfaz gráfica, inicializa el modo gráfico.
- En caso de que se haya inicializado el modo gráfico:
  - Permitirle al usuario definir qué es lo que quiere ver en la interfaz gráfica: los objetos, los Obstáculos-C, el mapa de carreteras, el robot, el área de movimientos permitidos, el área de trabajo y la ruta.
  - Monitorear al señalador cuando se haya definido una ruta para permitirle al usuario introducir la nueva meta para el robot.
  - Graficar todo lo que el usuario haya elegido.
- En caso de que se haya elegido la interfaz de texto:
  - Mostrarle al usuario las configuraciones que adquiere el robot
  - Cuando el robot ha alcanzado la meta o cuando ésta no es alcanzable, preguntarle al usuario cuál es la nueva meta.

Cuando se corre un programa en CLIPS, se introduce en la memoria de trabajo el hecho `(initial-fact)`. Si una regla se activa con éste hecho es posible garantizar que sea la primera en ejecutarse. Aprovechando ésto, se realizó una regla que hace todas las preguntas sobre el archivo del mapa y la configuración inicial del robot. Esta regla también introduce un conjunto de hechos que son los antecedentes para las reglas que inicializan las comunicaciones, el despliegue y la construcción del mapa de caminos.

La regla que inicializa las comunicaciones le pregunta al usuario si quiere simular los movimientos o si quiere ver que el robot los ejecute e introduce los hechos necesarios para la adecuada operación del programa en cualquiera de los dos casos. Más adelante se describe como se inicializa la comunicación con el robot.

La regla que inicializa el despliegue le pregunta al usuario si quiere usar una interfaz gráfica. Si el usuario contesta que no simplemente informa que no está abierta la interfaz gráfica. Si la respuesta es afirmativa, entonces se le pregunta que quiere ver desplegado: los objetos, las áreas prohibidas, los límites, el área permitida o el mapa de carreteras. El usuario pueder ver todo, sólo algunos elementos o ninguno (en cuyo caso sólo aparece el robot). También se le pide al usuario un valor de "zoom", en función del cuál se determina el tamaño de lo que aparece en pantalla.

Para desplegar un objeto en la pantalla, se utilizó un conjunto de funciones que previamente se le habían introducido a CLIPS y se realizaron otras para leer información del señalador o mouse. Este conjunto de funciones son las funciones CONDOR [21], y sirven para abrir una ventana, trazar líneas, círculos u objetos, y requieren que se cuente con las librerías de Motif para ser compiladas.

Para poder desplegar los objetos con diferentes tamaños se utiliza también el concepto de transformación de coordenadas, tratado en la sección 3.6.1 del capítulo 3.

Para recibir una órden del mouse se hizo una función que a su vez utiliza una de las funciones CONDOR para solicitar el estado del ratón. Esta función regresa el botón del ratón que haya sido oprimido estando dentro de la ventana de despliegue gráfico y sus coordenadas. Cuando el botón oprimido fué el central, termina el programa, cuando el botón fué el izquierdo, se le ordena al robot buscar un camino para llegar a la posición indicada por el señalador, si el botón fué el derecho, la orden es igual que con el botón izquierdo, pero además se redibuja todo el entorno gráfico.

5.1.5 Respuesta a solicitudes de información

El *planificador* recibe las solicitudes de información a través de un mecanismo de comunicación entre procesos. Éstas consisten en peticiones de calificación de una posición, que puede ir desde posición inalcanzable hasta posición excelente dependiendo de su cercanía con la meta. Este mecanismo tiene como finalidad que los otros módulos puedan contar con información útil para tomar decisiones.

5.2 Navegador

El navegador tiene como mision descomponer la ruta generada por el planificador en movimientos individuales para poder seguirla, y ordenar al piloto que realize esos movimientos. Para cumplir con su objetivo necesita:

- Separar la ruta indicada por el *planificador* en una secuencia de configuraciones parciales, identificando la posición de cada nodo en la secuencia.
- Si existe comunicación con el *piloto*, enviarle la siguiente configuración en la secuencia y esperar a que éste le indique la nueva configuración alcanzada, repitiendo hasta que se halla llegado a la configuración meta.

- Mostrar en la interfaz gráfica o de texto la secuencia de configuraciones generada y realizada (si hay comunicación con el piloto).

La ruta que genera el *planificador* está en una lista, por lo que para cumplir con su tarea, el *navegador* toma elemento por elemento de esa lista y si hay comunicación manda cada elemento al *piloto* y espera a que éste le reporte la configuración alcanzada. La lista de la ruta está en el hecho:

```
(path $?lista_de_nodos)
```

en donde `lista_de_nodos` está formada por la secuencia de nodos a alcanzar.

5.2.1 Identificación del siguiente nodo en la secuencia

La lista de nodos en la ruta está ordenada, por lo que simplemente se genera un conjunto de hechos `(target ?indice ?x ?y ?h)` donde `?indice` es el orden en que se debe alcanzar el punto, `?x` e `?y` son las coordenadas del punto a alcanzar, y `?h` es la orientación que debe adquirir el robot cuando llegue al punto destino. Por cada nodo dentro de la lista de nodos que forman la ruta debe existir un hecho con las características descritas, y el índice es consecutivo. El ángulo que se indica en la secuencia se obtiene usando la ecuación 3.13.

5.2.2 Ejecución de los movimientos para llegar a la siguiente configuración

Una vez que se definieron los hechos `target` para todos los nodos en la secuencia, se introduce el hecho `(execute target 1)` para que el robot alcanze el primer nodo. Si la comunicación con el robot está activada, entonces se pide al módulo de comunicaciones que le ordene al robot la siguiente configuración a alcanzar. Si no hay comunicaciones, se asume que el robot llegó a la siguiente configuración. A continuación se muestra la regla que produce este resultado:

```
(defrule send_order
 (declare (salience 1))
 (send_the_orders); hay que mandar las órdenes
 (communications ?state); estado de las comunicaciones
 (send_order_by_socket ?index); número de orden a enviar
 ;ya se envió la orden anterior
 (not (send_order_by_socket =(- ?index 1)))
 (robot ?name ?radius ?xr ?yr ?theta_r); posición del robot
 (target ?index ?xt ?yt ?headingt); configuración a alcanzar
 (client-id ?client)
 (server-id ?server)
=>
 (if (eq ?state enabled) then; comunicaciones habilitadas
  (bind ?data (str-cat "g " ?xt " " ?yt " " ?headingt))
  (printout t "order -> " ?data crlf)
```

```
  (CONDOR_send_data_client_network ?client ?data); manda órden
  (assert (read_socket)); indica leer el resultado
 else; no hay comunicaciones, entonces simula el resultado
  (assert (s2c ?index new_pos ?xt ?yt ?headingt)))
)
```

La manera en que se asume que el robot llegó a su posición, cuando no hay comunicaciones, es asertando el hecho `(s2c ?indice new_pos ?x ?y ?h)`. Ésto se hace para simular la respuesta que daría el piloto cuando el robot llegara a la configuración requerida. Cuando sí hay comunicaciones, se espera a que el piloto conteste que ya llegó a la siguiente configuración. Siempre que el piloto reporta que se alcanzó la posición, o que se simula este reporte, se actualiza la configuración del robot en el hecho `(robot ?nombre ?radio ?x ?y ?h)`.

5.2.3 Graficación de la ruta generada

Para desplegar la ruta que sigue el robot sólo se tiene que verificar si se abrió el ambiente gráfico y que se debe estar alcanzando la configuración (x, y, θ) y se hace el despliegue de la línea entre el punto en que se encuentra el robot y el punto de la configuración destino.

5.3 Piloto

El piloto se encarga de cumplir con las órdenes del navegador instruyendo al controlador los movimientos que tiene que realizar de la siguiente manera:

- Recibe del navegador la instrucción de alcanzar una configuración.
- Presenta el comportamiento de *alcanza configuración* que genera órdenes para mover al robot hacia la configuración solicitada.
- Presenta el comportamiento de *evade obstáculo* que genera órdenes para evadir obstáculos. Las órdenes las genera luego de consultar a una red neuronal y al planificador.
- Tiene un árbitro que decide a qué comportamiento le obedece el robot y le indica al controlador cuando y cuanto girar o trasladarse.
- Reporta al navegador cuando cumplió con la orden que se le dió y le informa la nueva configuración del robot.

Las comunicación con el navegador, con el planificador y con la red neuronal se realiza utilizando el mecanismo de comunicación entre procesos que se describe más adelante.

5.3.1 Comportamiento de alcanza configuración

El comportamiento de *alcanza configuración* calcula los movimientos necesarios para pasar de la posición en la que se encuentra el robot a la planteada por el navegador y emite instrucciones como: gira 20 grados o avanza .5 metros. Para realizar estos cálculos utiliza las ecuaciones 3.13 y 3.14.

Para generar las instrucciones de giro y desplazamiento, se sigue el siguiente algoritmo:

```
si el robot está detenido
 calcula giro para llegar a meta
 calcula desplazamiento para llegar a meta
 si el desplazamiento calculado es menor a tolerancia
  calcula giro para apuntar a orientación meta
 si el giro es mayor a una tolerancia
  da orden: ''gira giro''
 en otro caso
  si del desplazamiento calculado es menor a tolerancia
   da orden: ''desplaza desplazamiento''
 fin_si
fin_si
```

5.3.2 Comportamiento de evade obstáculo

El módulo de comunicaciones monitorea constantemente los sonares y las lecturas las transmite a la red neuronal que genera dos posibilidades de movimiento cuando se presente un objeto frente al robot, una de giro a la derecha y la otra de giro a la izquierda. El comportamiento *evadir obstáculo* utiliza las respuestas de la red neuronal para determinar hacia donde se debe mover el robot para que no choque, y mientras dure el temporizador de evasión, mantiene la última orden generada. A continuación se encuentra el algoritmo de éste comportamiento.

```
si hay red neuronal
 si el robot está avanzando
  si temporizador de evasión=0 y hay obstáculo
   calcula giro para evitar obstáculo
   calcula desplazamiento para evitar obstáculo
   establece temporizador de evasión en un Máximo
   da orden de evadir
 en_otro_caso
  si temporizador de evasión=0
   elimina orden de evadir con giro y desplazamiento calculados
 fin_si
fin_si
```

5.3.3 Árbitro

El árbitro le dá prioridad a las instrucciones generadas por el comportamiento de *evadir obstáculo.* La manera en que funciona se presenta a continuación:

```
Inicio
 si hay orden de evadir
  si el temporizador de evasión igual al Máximo
   gira ''giro de evadir''
   decrementa el temporizador de evasión
  en otro_caso
   si el temporizador de evasión igual al Máximo-1
    si el robot no está girando
     desplaza ''desplazamiento de evasión''
     decremento el temporizador de evasión
    fin_si
   en otro_caso
    si el temporizador de evasión es mayor a cero
     decrementa el temporizador de evasión
    en otro_caso
     para el robot
    fin_si
   fin_si
  fin_si
 en otro_caso
  si hay orden de girar
   gira ''giro''
  en otro_caso
   si hay orden de desplazarse
    desplaza ''desplazamiento''
  fin_si
 fin_si
fin
```

Todas las variables a las que se hace referencia fueron definidas en los algoritmos de los comportamientos *alcanza configuración* y *evade obstáculo.*

5.3.4 Scheduler

Como se mencionó anteriormente, los comportamientos descritos se implantan mediante un "scheduler". El "scheduler" permite definir en un programa un conjunto de funciones que se ejecuten periódicamente o después de que se reciba algún mensaje de la base.

Para hacer uso de este mecanismo se debe primero arrancar el programa servidor de comunicaciones (tcxServer), que permite comunicar a todos los programas que participan en el control del B14, y el servidor de la base (baseServer), que comunica a los motores y a los sensores con la aplicación del usuario.

En el programa del usuario debe localizarse al servidor de la base, introducir los módulos que participan en el scheduler y darle el control al scheduler. Ésto se hace con las siguientes instrucciones dentro de la función `main` del piloto:

```
registerBaseClient ( );
initClient("read_socket", commShutdown);
findBaseServer();/*se engancha al baseServer que está corriendo*/
RaiInit();       /*inicializa, no arranca, el scheduler*/
catchInterrupts();
initClientModules();
createModules ( );/*introduce comportamientos al scheduler*/
RaiStart();/*arranca el scheduler*/
return;
```

En la función `createModules` se inicializa la posición y orientación del robot, se establecen las velocidades, aceleraciones y torques de giros y traslaciones, y se informa al scheduler qué módulos tiene que ejecutar periódicamente (por poleo) y que módulos van a responder a un evento generado por registros de status o por la base (callbacks). A continuación se muestra el código de ésta función:

```
void createModules(void)
{
  RaiModule *communicateModule, *gotoModule;
  RaiModule *evadeModule, *arbitrateModule;
  printf ("Setting up the modules\n");
  loadPosition (0x80008000);   /* posición inicial */
  loadHeading (0);             /* orientación inicial */
  statusReportPeriod (100*256/1000); /* Tiempo entre reportes */
  registerStatusCallback (statusCallback);
  registerBaseCallback (baseCallback);
  setTranslateAcceleration (800);
  setTranslateVelocity (200);
  setRotateAcceleration (300);
  setRotateVelocity (300);
  setRotateTorque (255);
  /*Establece módulos para poleo cada 100 milisegundos */
  communicateModule = makeModule("communicate", modulesShutdown);
  addPolling (communicateModule, communicatePoll, 100);
  gotoModule = makeModule("goto", modulesShutdown);
  addPolling (gotoModule, gotoPoll, 50);
  evadeModule = makeModule("evade", modulesShutdown);
  addPolling (evadeModule, evadePoll, 50);
  arbitrateModule = makeModule("arbitrate", modulesShutdown);
  addPolling (arbitrateModule, arbitratePoll, 300);
  printf ("done\n");
}
```

El callback de status se activa cada que hay un cambio en el estado del robot, por ejemplo posición y orientación. Se le aprovecha para actualizar las variables de posición y orientación que utilizan los demás módulos cuando el odómetro reporta cambios, y es de hecho el que monitorea el odómetro.

El callback de la base se utiliza para detectar el momento en que el robot termina sus movimientos.

El módulo de comunicaciones es el que recibe las órdenes del navegador y le reporta los movimientos realizados, también se comunica con la red neuronal y lee sus salidas.

El módulo *goto* es el que tiene el comportamiento de *alcanza configuración*, el módulo *evade* es el que tiene el comportamiento de *evade obstáculo*, y el módulo *arbitrate* es el árbitro.

5.4 Red Neuronal

La Red Neuronal es el módulo que tiene las siguientes tareas:

- Recibir del piloto las lecturas de los sonares del robot.
- Introducir estas entradas a una red neuronal.
- La salida de la red neuronal se envía de regreso al piloto.

Las lecturas de los sonares son recibidas mediante el mecanismo de comunicación entre procesos que se describe en la sección 5.6.

Como se mencionó antes, la red neuronal que se utiliza es un perceptrón con algoritmo de entrenamiento de retropropagación. La razón para utilizar este tipo de red es que es una clase de redes neuronales ampliamente estudiada y que es un buén clasificador. Además, se encontró en Internet un programa para generar código en lenguaje C de una red neuronal de retropropagación con tan solo definir el número de capas de la misma, la cantidad de nodos en cada capa y la función de salida [2].

Se decidió realizar una red neuronal con tres capas. Tiene 8 entradas, en cada una de ellas se alimenta la lectura de uno de los sonares y 6 nodos de salida. La función de salida de cada neurona es la sigmoidal y entrega valores en el intervalo [0,1].

Para decidir el número de nodos en la capa oculta se utilizaron los siguientes criterios:

- El principal objetivo de un clasificador de patrones es lograr una velocidad aceptable en una clasificación correcta.
- El número de neuronas en la capa intermedia debe ser alto para que haya una mejor clasificación.

[2] En el apéndice B se puede consultar una breve introducción a Aspirin/MIGRAINES, el software de redes neuronales que se utilizó

- La relación entre el número de neuronas y el número de patrones de entrenamiento debe ser pequeña para acelerar el tiempo de entrenamiento.

La cantidad de nodos empleados se definió variable, para poder alterar fácilmente el número de neuronas en la capa oculta durante las pruebas.

El archivo que utiliza Aspirin/MIGRAINES para generar el programa red neuronal contiene lo siguiente:

```
DefineBlackBox evade /* Nombre de la red neuronal */
{
   OutputLayer-> Output /* Nombre de la capa de salida */
   InputSize-> 8 /* 8 líneas de entrada */
   Components->
   {
      PdpNode Output [6] /* 6 líneas de salida */
      {
         InputsFrom-> Hidden /* la salida recibe sus entradas
de la capa oculta*/
      }
      PdpNode Hidden [M] /* M neuronas en la capa oculta */
      {
         InputsFrom-> $INPUTS
      }
   }
}
```

que representa la red neuronal que se bosqueja en la figura 5.4.

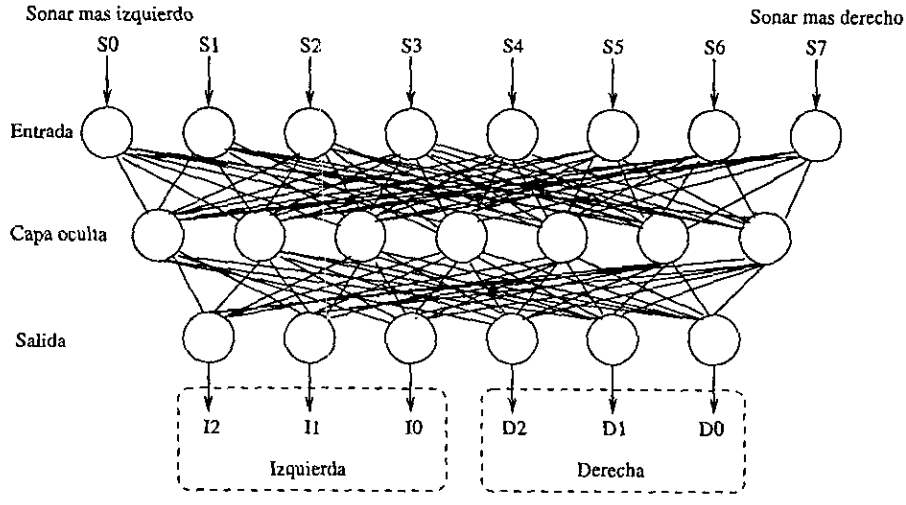


Figura 5.4: Estructura de la red neuronal utilizada

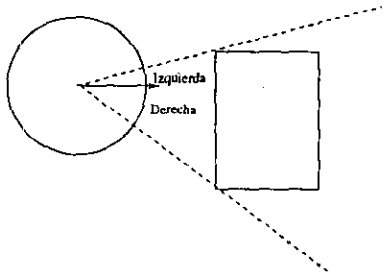


Figura 5.5: Interpretación de datos de la red neuronal

Se decidió agrupar las salidas de la red neuronal, a un grupo se le llamó izquierda y al otro derecha. Lo que en realidad se pretende es que la red neuronal informe de manera indirecta el espacio que ocupa un objeto que está frente a ella (figura 5.5). Para lograrlo se formaron dos grupos con las salidas de la red neuronal: el de la izquierda y el de la derecha. Aunque las salidas de la red neuronal están en el intervalo de [0,1], se decidió binarizarlas considerándolas como uno cuando superaran el valor de 0.5, y como 0 en caso contrario.

La razón para hacer esto fué tratar cada grupo de salidas de la red neuronal como un número binario único que indicara el espacio que ocupa el objeto frente al robot. Se definió que una salida 111 equivale a decir que hay un objeto obstruyendo el paso del robot frente al mismo y hacia el lado correspondiente al grupo al que pertenece el robot y que una salida 000 es equivalente a la ausencia de obstáculo en el mismo lado. Los valores intermedios son proporcionales al ángulo que el robot debe girar para no chocar con el obstáculo. Se decidió hacer que el robot gire $DI * G^o$, donde DI es el grupo de salidas de la red neuronal previamente binarizadas, G es una constante que hace que el robot gire 90^o cuando $DI = 111$.

Una véz que se genera el código en C de la red neuronal y que se compila, se procede a entrenarla. El entrenamiento se realiza a través de un archivo en donde están las entradas a la red y su salida ante las entradas mostradas. Para realizar el entrenamiento, se colocó al robot en las situaciones tipo que se muestran en la tabla 5.1 y se tomaron las lecturas de sus sonares, guardandolas en un archivo junto con la salida estimada para esas lecturas. En la segunda y la séptima situación el robot se colocó en su posición inicial, y se desplazó en incrementos de 15cm, en cada incremento se giró para tomar muestras cada 30^o de -60^o a 60^o. Entre la posición inicial y la final había 1.2m.

El archivo de entradas y salidas se procesó en la red neuronal para generar el archivo de pesos de la misma. Se trató que las situaciones de entrenamiento representaran las situaciones a las que se puede enfrentar el robot en su operación normal.

Una vez entrenada la red neuronal sólo se insertó en el programa que la comunica, que recibe una solicitud de respuesta, usa la función "system" para consultar a la red neuronal, y lee la salida de un archivo para regresarla por la

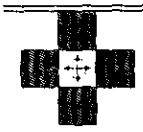
situación	*salida izq*	*salida der*	*muestras*	*comentario*
	111	111	5/30°	giros de 30 °
	000-111	000-111	25/15cm	5/30° de -60° a 60°
	111	111	20	una sola posición
	111	111	20	una sola posición
	111	111	20	una sola posición
	111-000-111	111-000-111	20 c/15°	de -105° a 105°
	000	000	25/15cm	5/30° de -60° a 60°
	111	111	5/10°	cajas a .4m

Tabla 5.1: Situaciones de entrenamiento de la red neuronal

red al programa que lo solicitó [3].

5.5 Controlador

El controlador se encarga de realizar los movimientos que indica el piloto, que pueden ser giros o desplazamientos. Para realizarlos controla directamente a los motores haciendo que cumplan con condiciones de posición, aceleración y torque preestablecidas. Éstas condiciones iniciales se fijaron en el programa del *piloto*.

El *controlador* ya estaba implementado en el robot B14, por lo tanto só-

[3]Este programa se lista completo en la sección 5.6

lo se necesitó incorporarlo y comunicarse con él. La plataforma de HW del controlador es el microcontrolador NEC 78310L.

5.6 Comunicación entre módulos

La comunicación entre los módulos (excepto entre el piloto y el controlador) se realiza mediante sockets, que es el mecanismo definido por UNIX BSD para comunicar procesos. Ésto permite que los módulos puedan operar en computadoras distintas, con el único requisito de que estén interconectadas a través de una red.

En el sistema operativo UNIX existen varias API's que permiten la comunicación entre procesos [22], las más importantes son los sockets de Berkeley y la Interfaz de la Capa de Transporte (TLI) de system V. Ambas utilizan los servicios de TCP/IP para realizar la comunicación, por lo que ya se han traducido a otras plataformas para permitir comunicación entre computadoras sin importar el sistema operativo con el que trabajen.

Las comunicación se realiza mediante un descriptor de archivo, y los procesos que necesitan comunicarse solo deben considerar lo siguiente:

- Existe una relación cliente-servidor no simétrica, en la que el programa que inicia una conexión debe saber que papel (cliente o servidor) está jugando.
- Hay dos tipos de conexiones: orientadas a conexión y sin conexión. En el primer tipo una vez que se abre una conexión con otro proceso, la I/O de red en esa conexión es siempre con el mismo proceso y el protocolo se asegura que los datos que un proceso envíe sean recibidos por el otro. En el segundo tipo, las operaciones de I/O pueden ser con distintos procesos en distintos hosts y el protocolo no verifica que los datos enviados sean recibidos.
- En ocasiones es necesario conocer el nombre del proceso con el que se realiza la comunicación, para verificar que tenga la autoridad para solicitar un servicio.
- Los parámetros que intervienen en una comunicación son: protocolo, dirección local, proceso local, dirección foranea, proceso foraneo.
- El sistema operativo UNIX mantiene comunicaciones orientadas a corrientes (stream) no a mensajes.
- Se deben soportar varios protocolos de comunicación. Dado esto no se pueden utilizar 32 bits para representar una dirección, porque si bien en Internet es adecuado, hay otros protocolos que utilizan direcciones con más bits.

Se eligió realizar la comunicación a través de sockets UDP, que es un protocolo sin conexión. Aunque éste protocolo no se asegura de que los datos enviados por un proceso sean recibidos por su destinatario, es más rápido que TCP y la desventaja de la ausencia de conexión no es un problema importante, ya que todos los procesos corren o en la misma computadora o en computadoras muy cercanas.

Ya que todos los programas realizan el mismo tipo de comunicación, aquí se muestra el de la red neuronal, que es el más pequeño y que muestra cómo se recibe información y se dán respuestas por la red.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "gsNetPoll.h"
#define INPUT_NAME "./input.agent.data"
#define OUTPUT_NAME "./output.agent"
void main(int argc, char **argv){
  char comando[256]="evade Network.Finished -d input.agent.df
    -E -p|grep Outputs|cut -d: -f2>output.agent";/*red neuronal*/
  char salida[256]="";
  char received[256]="", old_received[256]="";
  FILE *i_f, *o_f;
  GSNPS *reading, *writing, *open_network_conection();
  if (argc == 2) {
    writing = open_network_conection (argv[1], 2002, 256, "w");
    reading = open_network_conection ("", 2003, 255, "r");
    if (writing == NULL || reading == NULL){
      printf ("communications error\n");
      return;}
    else
      printf ("sockets opened\n");}
  else {
    printf ("use:neural_network host\n");
    return;}
  while (strcmp (received, "q") != 0){
    strcpy (received, receive_data_client_network (reading));
    if (strcmp (received, old_received)){
      printf("\nreceived data ->%s", received);
      i_f = fopen (INPUT_NAME, "w");
      fprintf (i_f, "%s\n 0 0 0 0 0 0\n", received);
      fclose (i_f);
      printf("consulta a la red neuronal ...\n");
      system(comando);
      o_f = fopen (OUTPUT_NAME, "r");
      fgets (salida, 200, o_f);
      fclose (o_f);
```

```
      printf("resultado: %s", salida);
      send_data_client_network (writing, salida);}
    strcpy (old_received, received);}
  close_network_conection (reading);
  close_network_conection (writing);
}
```

Para hacer la comunicación entre el *piloto* y el *controlador*, se utilizó la arquitectura Cliente-Servidor de Beesoft, formada por un conjunto de programas que tiene incorporados el robot B14 y que tienen varias funciones. Para el problema de éste trabajo se utiliza el programa "baseServer" y "tcxServer". "baseServer" envía órdenes a los controladores de los motores para que se puedan realizar desplazamientos y giros, además recibe las lecturas de los sensores del robot. "tcxServer" coordina las comunicaciones entre todas las computadoras del robot y sus dispositivos así como entre los servidores y los programas de usuario (clientes). Existen otros programas de propósito específico, como un servidor de voz y otro para usar un brazo manipulador, pero no se utilizaron. En la figura 5.6 se bosqueja esta arquitectura [19].

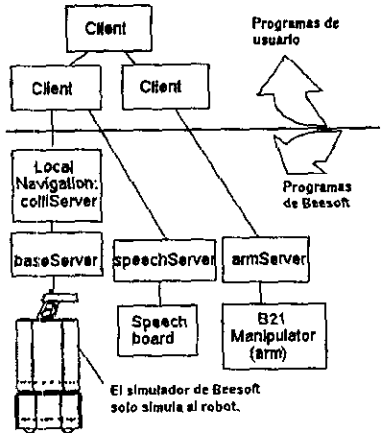


Figura 5.6: Arquitectura Cliente-Servidor de Beesoft

5.7 Integración de los módulos

Una vez que se construyeron todos los módulos sólo es necesario que operen en conjunto. El planificador y el navegador se implementaron en CLIPS, en el que un script es el encargado de cargar las reglas de ambos y ejecutarlas. Las reglas están en varios archivos, por lo que el script tiene que cargarlos todos. El contenido de este script es;

```
(load functions.clp);       las funciones
```

```
(load initial_rules.clp);  reglas iniciales
(load draw.clp);     reglas de dibujo
(load make_nodes.clp);     construcción del roadmap
(load mode_of_search.clp); definición del modo de búsqueda
(load tree.clp);     construcción del árbol
(load navigator.clp);     navegador
(reset)
(run)
```

Se puede ver que el script carga siete archivos. Los primeros seis corresponden al planificador y el último al navegador.

El piloto está codificado en un único programa que hace uso del API del B14 y que debe correr en la máquina Pentium que éste tiene. El único programa que forzosamente debe correr en el B14 es el piloto, todos los demás pueden correr en cualquier computadora que esté conectada a través de una red TCP/IP con el robot, aunque lo ideal es que todos los programas corran en el mismo. Hay dos maneras de ejecutar el programa del piloto: cuando hace uso de la red neuronal y cuando no la utiliza.

Para ejecutar el piloto cuando usa la red neuronal para evadir obstáculos se ejecuta de la siguiente manera:

```
%pilot host_del_navegador host_de_la_red_neuronal
```

donde `host_del_navegador` es la dirección IP de la computadora donde se ejecuta el programa del navegador, y `host_de_la_red_neuronal` es, a su vez, la dirección IP de la máquina donde se ejecuta la simulación de la red neuronal.

Para ejecutar el piloto sin que use la red neuronal basta con ejecutar su programa sin el parámetro que indica el host de la red neuronal. En este caso el comportamiento de *evadir obstáculo* no participa en la toma de decisiones para mover al robot.

Un aspecto que se debe cuidar al integrar los módulos es verificar que los programas se comunican a través del puerto correspondiente. La red neuronal recibe datos a través del puerto 2003 y entrega sus resultados por el puerto 2002. El navegador envía órdenes al piloto por el puerto 2000, y lee sus resultados y solicitudes por el puerto 2001. El piloto por su parte utiliza las direcciones del navegador y de la red neuronal para comunicarse con ellos.

Capítulo 6

Experimentos y resultados

En éste capítulo se describen las pruebas realizadas para verificar el funcionamiento del robot. Primero se probaron los módulos por separado y al final se probaron integrados. Todos los experimentos se realizaron en el Laboratorio de Interfaces Intelientes del edificio Valdez Vallejo de la Facultad de Ingeniería.

Se introdujeron nuevos elementos al área de trabajo del robot de manera que el mapa del planificador quedó como se observa en la figura 6.1. Para hacer las pruebas se definió una aplicación que se le puede dar al robot: Hay un conjunto de depósitos que el robot debe visitar para llevar objetos entre ellos.

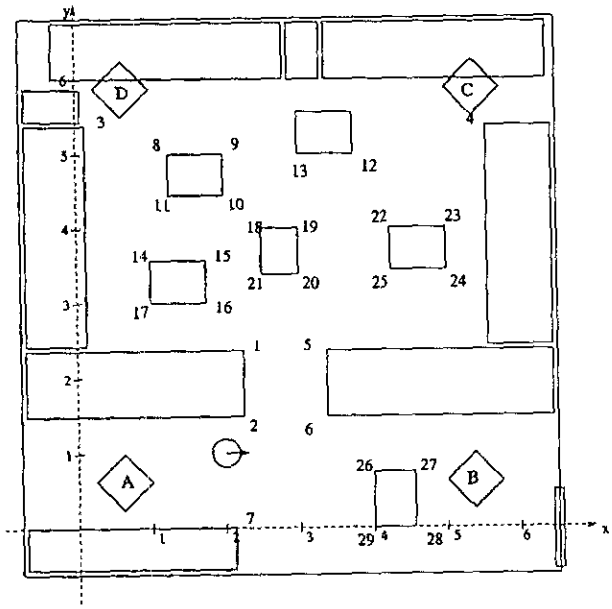


Figura 6.1: Mapa del Laboratorio de Interfaces Inteligentes

Se definieron cuatro depósitos que en la figura 6.1 están etiquetados como A, B, C y D. Para llegar a los depósitos el robot tiene que alcanzar las configuraciones que se muestran en la tabla 6 partiendo inicialmente de la configuración denominada como O, que en la figura es la posición del robot representado como un círculo con una flecha que indica su orientación.

depósito	x	y	*orientación*
A	1	1	-135^o
B	5	1	-45^0
C	5	5.5	45^o
D	1	5.5	135^o
O	2	1	0^o

Se generaron, de manera aleatoria, diez trayectorias para que el robot visitara primero un depósito y luego otro simulando el transporte de algún objeto. Estas trayectorias se utilizan a lo largo de los siguientes experimentos.

6.1 Planificador

En principio se probó el desempeño del planificador de manera independiente, para lo que al ejecutar el programa de planificador/navegador se especificó que se realizaría sólo una simulación. Ésto con el fin de evitar que el robot hiciera los movimientos. Se midieron los siguientes aspectos:

Tiempo de Generación de la Ruta (TGR) es el tiempo que ocupó el planificador en generar la ruta para llegar al destino medido en segundos.

Longitud de la ruta (LR) es la longitud de la ruta encontrada por el planificador medida en metros.

Ruta encontrada secuencia de nodos en el mapa de carreteras visitadas al seguir la ruta encontrada. En la figura 6.1 las posiciones aproximadas de los nodos están etiquetadas con números más grandes.

Longitud de la ruta más corta (LRMC) es la longitud de la ruta más corta para llegar al destino medida en metros.

Mejor Ruta secuencia de nodos que conforman la mejor ruta para llegar al destino.

%Error (%E) es el resultado de dividir la diferencia entre LR y LRMC entre LRMC y multiplicar el resultado por 100.

La prueba consistió en alimentar al robot las trayectorias generadas y medir los resultados. En la tabla 6.1 se muestran los resultados obtenidos. Por ejemplo, para que el robot lleve un objeto del depósito D al depósito B, como se observa

Ruta	*Meta*	*TGR*	*LR*	*LRMC*	*%E*	*Ruta Encontrada* Mejor Ruta
D->B	D	65	9.3	5.6	67	O->29->1->15->14->D O->2->1->15->11->8->D
	B	74	7.2	7.2	0	D->9->21->6->B D->9->21->6->B
D->A	D	54	7.2	7.2	0	B->6->21->9->D B->6->21->9->D
	A	90	7.5	6.8	11	D->9->21->6->A D->9->21->1->2->A
B->D	B	18	4.1	4	2	A->2->B A->26->27->B
	D	52	7.2	7.2	0	B->6->21->9->D B->6->21->9->D
B->C	B	72	7.2	7.2	0	D->9->21->6->B D->9->21->6->B
	C	18	7.5	6.6	13	B->2->12->C B->6->5->22->C
A->C	A	19	6.5	6.4	2	C->12->2->A C->22->5->2->A
	C	18	6.5	6.4	2	A->2->12->C A->2->5->22->C
A->D	A	19	6.5	6.4	2	C->12->2->A C->22->5->2->A
	D	61	10.2	6.5	57	A->29->1->15->14->D A->2->1->15->11->8->D
A->B	A	90	7.5	6.8	11	D->9->21->6->A D->9->21->1->2->A
	B	19	4.1	4	2	A->2->B A->26->27->B
D->A	D	51	7.2	7.2	0	B->6->21->9->D B->6->21->9->D
	A	90	7.5	6.8	11	D->9->21->6->A D->9->21->1->2->A
C->D	C	18	6.5	6.4	2	A->2->12->C A->2->5->22->C
	D	22	4.8	4.4	10	C->19->9->D C->12->13->9->D
C->B	C	19	4.8	4.4	10	D->9->19->C D->9->13->12->C
	B	19	7.5	6.6	13	C->12->2->B C->22->5->6->B
prom		43	6.8	6.2	10.7	

Tabla 6.1: Pruebas del planificador

en el primer experimento, el robot primero tiene que alcanzar D y luego B. Cuando se le planteó al robot llegar a D, el planificador tardó 65 segundos en generar la ruta, la longitud de la ruta fúe de 9.3 m, mientras que la ruta más corta fué de sólo 5.6 m, por lo que la ruta encontrada fué 67 % más larga que la más corta. La ruta encontrada implicó que el robot, partiendo del nodo O, visitara en órden los nodos: 29, 1, 15, 14 y D. Ésta ruta aunque no es la más corta, si es la que ménos móvimientos involucra, ya que en la más corta hay que visitar más nodos que són: el 2, el 1, el 15, el 11, el 8 y la meta D. Las etiquetas en los nodos corresponden a los números mostrados en la figura 6.1.

Al revisar los resultados se puede encontrar que el valor máximo de %Error es de 67%, pero que en todos los casos en que el planificador no encuentra el camino más corto, sí hace una cantidad de movimientos menor, lo que para desplazamientos cortos resulta en un ahorro de energía.

6.2 Navegador

El navegador está integrado en el sistema experto y las pruebas del planificador utilizan la parte de simulación del navegador. Las pruebas hechas en el planificador permitieron observar en la interfaz gráfica los movimientos que debía hacer el robot, incluyendo cambios de posición y desplazamientos. El hecho de poder dar seguimiento visual a estos movimientos indica que el navegador funciona correctamente.

6.3 Piloto

Al piloto se le hicieron dos pruebas, la primera incluyó únicamente al comportamiento *Alcanza configuración*, la segunda fué con los dos comportamientos funcionando. Esto permite evaluar por separado la funcionalidad de la red neuronal.

6.3.1 Alcanza configuración

Este comportamiento está compuesto de dos mecanismos, el que orienta al robot hacia la configuración destino y el que lo hace caminar hacia la misma. Ambos mecanismos se definieron a su vez como comportamientos.

Se notó que al operar en conjunto, siempre llegan a la configuración destino con un error de $\pm 5\%$. Este error depende principalmente de una adecuada calibración del sistema mecánico de la base, por lo que el dato puede variar. En las secciones que siguen se describen varios experimentos que consideran este aspecto.

Para llegar a su meta, el robot no siempre realiza la misma secuencia de movimientos, ya que los dos comportamientos son asíncronos y ésto provoca que en ocasiones uno de los comportamientos se empiece a ejecutar cuando el otro aún no termina. Es por ésto que a veces se observa que el robot gira varias veces antes de orientarse hacia su destino.

Otro comportamiento que presenta el robot es que en ocasiones no llega a su meta con una única combinación de giro y desplazamiento. Si por ejemplo, al orientarse hacia su destino quedó con una orientación ligeramente distinta a la que debió adquirir y el desplazamiento fué muy grande, el robot no llega al destino en el primer movimiento, por lo que sus comportamientos de orientarse y desplazarse se vuelven a activar para llegar a su destino.

6.3.2 Alcanza configuración + evade obstáculo

Origen (x,y,θ)	*Meta (x,y,θ)*	*NC*	*GI*	TR	NO	COR	¿Llegó?
(2.6,3.1,180)	(0.5,5,0)	4	13	105	2	si	si
(0.5,5,0)	(5,5,180)	4	15	150	2	si	si
(6.35,4.3,150)	(3,3,0)	1	6	90	1	no	si
(2.6,2.1,30)	(0.5,3,0)	1	2	45	1	si	si
(0.6,2.9,0)	(5,5,180)	4	1	60	3	si	si
(5.17,5.3,180)	(1.5,4,0)	5	10	50	1	no	si
(1.9,3.6,30)	(3.5,3.5,180)	0	1	34	1	si	si
(3.5,3,0)	(2.5,5.5,-90)	3	3	50	1	si	si
(2.5,5.65,-90)	(2.25,3.25,90)	3	20	79	2	no	no
(3.25,2.25,90)	(0.5,5.5,-90)	2	18	90	1	no	no

Tabla 6.2: Pruebas del piloto con red neuronal

Para probar al robot evitando obstáculos mientras alcanza su destino, se le indicó que llegara a distintas metas sin darle información sobre su entorno, de modo que no tenía ningún conocimiento de los obstáculos que estaban entre él y su meta. Después se hizo una función booleana para medir el desempeño de la red neuronal y se volvió a hacer la prueba. Los aspectos que se midieron son los siguientes:

Número de choques (NC) Cantidad de veces que chocó el robot con un objeto en su camino.

Giros inecesarios (GI) Número de veces que el robot giró sin que hubiera un objeto delante de él. Esta es una forma indirecta de medir la cantidad de veces que el robot detectó un obstáculo cuando no lo había.

Tiempo de recorrido (TR) tiempo que el robot tardó en llegar a su meta medido en segundos.

Número de obstáculos (NO) Cantidad de obstáculos que se interponían entre el robot y la meta trazando una línea recta desde su posición inicial hasta la meta.

Corrección (COR) indica si fué necesario corregir la posición del robot una vez que este reporto haber alcanzado la meta. Esta es una forma indirecta de evaluar al odómetro del robot.

En la tabla 6.2 se muestra el resumen de los resultados obtenidos cuando el piloto empleó la red neuronal para evadir los obstáculos. Por ejemplo, en el primer caso el robot se encontraba inicialmente en las coordenadas (2.6, 3.1) con una orientación de 180^0 y se le planteó llegar a las coordenadas (0.5, 5.0) con una orientación de 0^0. Al tratar de cumplir con la orden chocó cuatro veces y giró sin haber obstáculo en medio 13 veces, el tiempo de recorrido fué de 105 segundos y superó dos obstáculos. Al final de su recorrido fué necesario corregir sus coordenadas, ya que el error era muy grande. En los dos últimos casos el robot no llegó a su destino mientras no se quitaron los obstáculos.

Verificación de la red neuronal

Origen (x,y,θ)	*Meta (x,y,θ)*	*NC*	*GI*	TR	NO	COR	¿Llegó?
(0,0,0)	(2.5,0,0)	-	-	-	1	si	no
(2.5,0,0)	(0,0,0)	5	10	90	1	si	si
(0,0,0)	(0,2.5,180)	-	-	-	-	si	no
(0,2.5,180)	(0,0,180)	1	1	36	1	si	si
(0,0,180)	(-2.5,0,90)	2	1	40	1	no	si
(-2.5,0,90)	(0,0,0)	-	-	-	1	si	no
(0,0,0)	(0,-2.5,90)	2	3	40	1	si	si
(0,-2.5,90)	(0,0,-90)	1	2	25	1	no	si
(0,0,-90)	(2.5,0,90)	-	-	-	1	si	no
(2.5,0,90)	(0,0,0)	1	4	60	1	si	si

Tabla 6.3: Pruebas del piloto con función booleana

Se hizo una prueba adicional con el fin de evaluar la utilidad de la red neuronal. Se hizo una función con las mismas entradas y salidas que la red neuronal, tratando de que resolviera el mismo problema. Esta función binariza las entradas y calcula las seis salidas con funciones booleanas. Ésto permitió contar con una función similar a la red neuronal que se probó de la misma forma. Los resultados de estas pruebas se muestran en la tabla 6.3. Se puede notar que en esta prueba sólo se le interpuso un obstáculo al robot, y que en tres casos el robot no pudo llegar a su destino, en la mayoría de los casos se necesitó corregir las coordenadas del robot por las mismas razones que en el experimento anterior.

Los resultados obtenidos no son concluyentes, aparentemente la red neuronal es un poco mejor que las funciones booleanas, ya que en cuatro ocasiones el robot no llegó a su destino, a pesar de que en dos ocasiones los obstáculos se movieron una vez que chocó con ellos para facilitarle la llegada a su meta.

Ruta	*Meta*	*Punto Alcanzado*	*COR*	*TR*
D->B	D	(1.25,5.74)	si	77
	B	(4.78,0.85)	si	60
D->A	D	(0.6,5.17)	si	55
	A	(1,1.17)	si	55
B->D	B	(5,1.17)	si	20
	D	(0.5,5)	si	60
B->C	B	(4.8,5.8)	si	49
	C	(5.15,5.85)	si	40
A->C	A	(0.9,1)	no	40
	C	(4,5.5)	si	40
A->D	A	(0.9,1.15)	si	41
	D	(0.6,5.85)	si	90
A->B	A	(1,1.3)	si	55
	B	(5,1.5)	si	25
D->A	D	(0.8,5.8)	si	54
	A	(1,1.15)	no	59
C->D	C	(4.5,5.7)	no	60
	D	(1.1,6.1)	si	35
C->B	C	(5.15,5.8)	si	35
	B	(5.1,1.15)	no	54

Tabla 6.4: pruebas de todos los módulos sin evasión de obstáculos

6.4 Integración de los elementos

Se hicieron dos pruebas de integración con todos los módulos del robot operando en conjunto, la primera excluyó el comportamiento de *evade obstáculo* y la segunda fué una prueba con todos los comportamientos. En ambas pruebas se siguieron las mismas metas que para la prueba del planificador, por lo que las tablas con los resultados son una ampliación de la tabla 6.1

En la primera prueba (sin el comportamiento de *evade obstáculo* sólo se midieron las coordenadas alcanzadas por el robot y el tiempo que tardó en su recorrido y si necesitó o no corrección en cada una de ellas. Los resultados se muestran en la tabla 6.4. Por ejemplo, en el primer caso, el robot debió alcanzar el depósito D en las coordenadas (1,5.5), sin embargo llegó a las coordenadas (1.25, 5.74), por lo que se necesitó corregir su posición. El tiempo que tardó en llegar a la meta es de 77 segundos.

En los resultados de esta prueba se nota que en la mayoría de los casos el robot necesitó corrección de posición a pesar de no estar demasiado lejos de la posición destino, esto se debe a problemas mecánicos que se sufrieron durante esta etapa, además de que los encodificadores nunca serán 100% confiables.

En la segunda prueba, se midieron de nueva cuenta los aspectos medidos

en las pruebas del navegador (NC, GI, TR, NO y COR) y se incluyeron las coordenadas que se le tuvieron que indicar al robot cuando se necesitó que corrigiera su posición.

Ruta	*Meta*	*NC*	*GI*	*TR*	*NO*	*COR*
D->B	D	6	4	120	2	si(2.15,6.15,35)
	B	3	20	240	2	si(5,1.5,-45)
D->A	D	4	40	300	3	si(0.3,4.1,90)
	A	2	2	120	2	si(1.5,1,-135)
B->D	B	2	1	10	105	si(5,1.4,-45)
	D	4	8	125	3	si(1.15,5.4,-35)
B->C	B	2	4	158	4	si(4.7,0.4,-135)
	C	5	20	360	3	si(4.2,4.9,45)
A->C	A	2	5	130	1	si(1.6,0.7,-135)
	C	2	10	195	1	si(5,4,45)
A->D	A	7	20	240	2	si(2,1,0)
	D	1	2	120	3	si(1.3,5.4,135)
A->B	A	2	4	100	4	si(0.8,1.9,-135)
	B	2	4	135	1	no
D->A	D	—	—	—	—	—
	A	3	5	162	3	si(2.5,1.5,-135)
C->D	C	4	10	177	2	si((5.5,4,90)
	D	0	0	45	2	no
C->B	C	2	10	133	2	si(4.5,5.5,135)
	B	1	5	170	3	no

Tabla 6.5: pruebas con todos los módulos funcionando

Estas pruebas muestran que el enfoque de comportamientos y el tradicional se pueden integrar en un sólo entorno logrando un efecto positivo en el desempeño. También se pudo observar que se depende demasiado del odómetro del robot, si éste falla el robot no llega a la meta planteada. Aún teniendo muy bién calibrados todos los sensores, el robot falla más de lo que se quisiera.

Capítulo 7

Conclusiones

En los siguientes párrafos se detallan las principales conclusiones que se obtuvieron como resultado de este trabajo.

El sistema experto

Cuando se implementaron el planificador y el navegador se encontró que el sistema experto ofrece varias ventajas. La separación explícita entre los datos y los conocimientos permite hacer crecer el sistema de manera modular, posibilitando agregar conocimientos sobre planeación y navegación sin necesidad de alterar en lo fundamental el sistema que ya opera. También facilita la construcción del mapa del entorno del robot, ya que sólo es necesario agragar o eliminar líneas de un archivo en donde éste se define.

Otra ventaja es que si se desea cambiar el modo de operación de cualquiera de los *módulos de concocimientos* basta con sustituirlo. Esta es una forma de cambiar las políticas de la búsqueda para que el criterio de minimización sea otro. Por ejemplo, si se quiere que el árbol de búsqueda se realize de otra manera, sólo se sustituye el módulo de creación del árbol por otro que contenga las reglas apropiadas. O si por ejemplo, se necesita utilizar otro paradigma de planificación, como el de campos potenciales, se traduce a reglas de producción y se pone en lugar del árbol de búsqueda.

El planificador

Los resultados que se obtuvieron con el planificador siempre fueron adecuados. La ruta generada, aunque no siempre fué la más corta, si minimizó la cantidad de movimientos. Ésto es importante ya que se puede reducir la energía consumida por el robot, aunque sólo cuando el entorno de trabajo no es muy grande y cuando los movimientos del robot son contínuos. Si los movimientos fueran contínuos y el robot avanzaray girara simultaneamente de forma que el robot no se detuviera por completo antes de llegar a la meta, el ahorro de energía probablemente no sería tanto.

Por otro lado, la distancia a recorrer por el robot siempre fué razonable. Lo más alejado que estuvo de la ruta más corta fué en 60%, y aunque ésto puede parecer mucho, no lo es tanto si se considera que en la mayoría de los movimientos no excedió en más de 10% a la ruta más corta.

La red neuronal

Se propuso evaluar una red neuronal para lidiar con las imprecisiones del robot y actuar como un filtro que al mismo tiempo transformara las lecturas de los sonares en "opiniones" para indicar la ubicación de los obstáculos. Al implantar la red neuronal en un software previamente desarrollado se obtuvo flexibilidad. De manera que modificar la estructura de la red neuronal sólo implicó cambiar el archivo de definición y volver a compilar.

El principal inconveniente de la red neuronal fué que conforme se utilizaban distintos conjuntos de entrenamiento, el desempeño del robot cambió, en ocasiones para bién, en otras para mal. No se encontró en la literatura disponible un criterio que guiara la elección del conjunto de entrenamiento y ésto representó una gran carencia.

Los resultados de las pruebas de la red neuronal reportan una cantidad elevada de giros innecesarios, ésto es una prueba indirecta de falsos positivos, es decir, el robot está detectando obstáculos en muchas ocasiones en que no los hay. También se presentan muchos falsos negativos, es decir no se detecta la presencia de obstáculos reales.

Al comparar los resultados entregados por la red neuronal con los que entregaron las funciones booleanas, se encontró que la red neuronal es ligeramente mejor que las funciones booleanas, pero no de manera significativa, por lo que no se puede concluir que las redes neuronales, utilizándolas de esta forma, aporten demasiado. De cualquier forma, se propone hacer muchas más pruebas para determinar con precisión la aportación de éstas.

Se propone también evaluar las redes neuronales utilizando otro tipo de sensores y con otras funciones.

El piloto

El implantar el piloto usando el enfoque de control de robots basado en comportamientos, respondió a las expectativas, a pesar de los problemas que se tuvieron durante la implantación.

El problema más importante que se enfrentó fué la imprecisión del odómetro. Esto se nota en las pruebas del piloto, en las que se observa que se corrigió la posición del robot con mucha frecuencia. La razón es que el piloto utilizaba exclusivamente las lecturas del odómetro para determinar la posición del robot.

A pesar de que este problema se puede minimizar recalibrando los elementos mecánicos del robot, se propone agregar al robot un sensor de posición independiente del odómetro. De lo contrario no se puede contar con un robot que opere de manera confiable durante largos períodos de tiempo.

Otra solución puede ser no depender de la posición absoluta del robot, sino implantar mecanismos que permitan conocer la posición relativa del robot con respecto a referencias en su entorno. Las referencias se pueden obtener a través de conjuntos diversos detransductores, que pueden ser devisión, sonido, tacto u otros.

El enfoque de control mixto

Los distintos enfoques de control de robots no son necesariamente contradictorios, pues se pueden aprovechar las ventajas de ambos. El requisito para lograr una integración funcional es definir adecuadamente el protocolo de comunicación entre los distintos componentes del robot.

Es posible aprovechar la integración de los dos enfoques de control para fortalecer el comportamiento global del robot. Una manera es utilizar el planificador para supervise el entrenamiento del piloto, en el cual se pueden utilizar más sensores. Ésto se haría dejando que el piloto comienze a realizar movimientos en su entorno y el planificador, que debe contar con un mapa confiable, supervisa los movimientos "premiando" al piloto cuando acierta y "castigándolo" cuando falla.

Otra forma de aprovechar la integración entre los dos enfoques es que el piloto informe al planificador las lecturas de sus sensores para que éste confirme o descarte la información de su mapa. Ésto se podría aprovechar incluso para construir el mapa sin necesidad de la asistencia de un operador.

Al final de éste trabajo, todavía no se cuenta con suficientes elementos para que el robot opere en un entorno natural, pero ya se cuenta con una base de desarrollo. Un trabajo que se propone desarrollar es la integración de los sistemas desarrollados en éste con mecanismos de visión para la detección de objetos como apoyo a la navegación del robot.

Apéndice A

CLIPS

CLIPS es un shell para realizar sistemas expertos que se empezó a desarrollar en el centro espacial Johnson de la NASA desde 1985 y que ya está en su versión 6.0.

A.1 interacción con CLIPS

CLIPS es un intérprete, así que los programas escritos para él se pueden alimentar mediante el teclado o a través de un archivo. En UNIX el comando para ejecutar el intérprete es:

```
\%clips [-f <archivo>]
```

Si la opción `-f` no se alimenta, el programa entra en su modo interactivo y admite instrucciones del usuario. Si por otro lado la opción `-f` se introduce, el programa busca el `archivo` y lo ejecuta.

A.2 Elementos básicos de programación

CLIPS admite ocho tipos de datos: float, integer, symbol, string, external-address, fact-address, instance-name e instance-address. Ninguna variable tiene que declararse, ya que CLIPS asigna el tipo en la primera asignación recibida.

A.2.1 Representación de datos

El mecanismo que más se utiliza en CLIPS para representar la información son los hechos. Los hechos se almacenan en una lista que también se conoce como base de hechos y forman la unidad fundamental de datos utilizada en las reglas. Las operaciones que se les pueden aplicar son: agregar (`assert`), retirar (`retract`), modificar (`modify`) y duplicar (`duplicate`). Estas operaciones se realizan dentro de las reglas o a través de interacción directa con el usuario.

Un hecho puede tener dos formas: ordenado o no ordenado. Un hecho ordenado es un simbolo seguido de cero o más campos separados por blancos y delimitado por paréntesis como en el caso de:

```
(robot uno 1 1 10)
(objeto caja 1 1 1 2 2 2 2 1)
```

Un hecho no ordenado se forma mediante la sentencia `deftemplate` que permite asociar a un dato un conjunto de slots o campos que pueden tener algún valor, cada campo es otro hecho. Por ejemplo:

```
(robot (posicion 1 1) (orientacion 10) (color azul))
```

El orden de los slots no tiene importancia.

En CLIPS los conocimientos se representan principalmente mediante reglas del tipo *si condicionantes, entonces consecuentes.* Estas reglas se escriben

A.2.2 Representación de conocimientos

Hay dos formas de representar el conocimiento. Los conocimientos heurísticos se representan mediante reglas de producción y los conocimientos procedurales mediante funciones.

Las reglas de producción se definen mediante la construcción `defrule` que tiene la siguiente sintaxis:

```
(defrule <nombre_de_regla> [<comentario>]
   [<declaración>]  ; Propiedades de la regla
   <elemento_condicional>* ; Lado izquierdo de la regla
 =>
   <acción>*) ; Lado derecho de la regla
```

los campos rodeados por corchetes son opcionales. Para incluir un comentario se introduce un ";" y CLIPS toma el resto de la línea como comentario. Los eloementos condicionales y la acción pueden ser cero o más. Las acciones son una secuencia de llamadas a funciones, afirmaciones o eliminación de hechos. Cuando se cumplen todos los elementos condicionales se ejecutan en orden todas las acciones.

Las funciones se definen con la construcción `deffunction` cuya sintaxis se muestra a continuación:

```
(deffunction <nombre_de_función> [<comentario>]
   (<parámetro regular>* [<parámetro comodín>])
   <acción>*)
```

Un parámetro regular es una variable de un sólo campo que se reconoce por estar antecedida por un signo de interrogación ?, mientras el parámetro comodín es una variable multicampo que se reconoce por estar antecedida de un signo de pesos seguido de la interrogación $?.

El resultado de la función es el resultado de la última acción desarrollada en la misma.

CLIPS se puede encontrar en Internet en `http://www.fi-b.unam.mx/soft.html`.

Apéndice B

Aspirin/MIGRAINES

Aspirin/MIGRAINES es un programa de dominio público cuyo propósito es simplificar y acelerar la investigación en redes neuronales y facilitar la ejecución de sistemas de redes neuronales. Está formado por dos capas: Aspirin y MIGRAINES.

Aspirin es un lenguaje declarativo para describir redes neuronales complejas de manera simple. La descripción de la red neuronal en Aspirin se usa para generar un programa que simula la red descrita.

MIGRAINES es una interfaz para evaluar e interactuar con la simulación de la red neuronal descrita en Aspirin.

B.1 Desarrollo de redes neuronales artificiales

El proceso de desarrollo de la red neuronal es como sigue:

1. Se define la estructura de la red neuronal.
2. Se describe la red neuronal en el lenguaje de Aspirin dentro de un archivo con extensión `.aspirin`.
3. Se compila la red neuronal.
4. El resultado es un programa en C con la red neuronal y un ejecutable que permite simular la red neuronal de manera interactiva (utilizando MIGRAINES) o a través de la línea de comandos.

B.2 Archivo de definición en Aspirin

El archivo de definición de la red neuronal puede ser como el que se muestra a continuación:

```
DefineBlackBox evade /* nombre de la red neuronal */
{
  OutputLayer-> Output
  InputSize-> 8 /* 8 entradas */
  Components->{
    PdpNode Output [6] /* 6 salidas */{
      InputsFrom-> Hidden /* sus entradas provienen de las*/
                          /* salidas de la capa oculta*/ }
    PdpNode Hidden [7] /* 7 neuronas en la capa oculta */{
      InputsFrom-> $INPUTS /* las entradas de la capa oculta
                  vienen de las entradas */}}
}
```

Este archivo describe a una red neuronal llamada **evade** que tiene 8 entradas, 7 neuronas ocultas y 6 salidas. Se puede ver que la definición es similar a la de un programa en C, pero utilizando estructuras propias de las redes neuronales.

B.3 Compilación y entrenamiento

Para compilar un archivo de definición es necesario tener instalado el paquete, copiar un archivo Makefile y ejecutar el comando de UNIX make. El resultado es un conjunto de archivos entre los que se encuentra un programa con la simulación de la red neuronal que ya se puede entrenar.

El entrenamiento se hace una vez que se cuenta con el programa ejecutable. Para realizarlo se necesita un archivo con extensión `.data` en donde se encuentren las entradas y las salidas correspondientes como se muestra a continuación:

```
/* primer conjunto */
0.097 0.120 0.201 0.093 0.093 1.000 0.226 0.093 /* Entrada */
1 1 1 1 1 1 /* Salida */

/* segundo conjunto */
0.097 0.121 0.198 0.093 0.093 1.000 0.231 0.093 /* Entrada */
1 1 1 1 1 1 /* Salida */
```

En este archivo sólo aparecen dos datos de entrenamiento, pero pueden aparecer tantos como se considere necesario. Durante el entrenamiento la red neuronal procesa todos los datos que se le alimentan tratando de que sus salidas se ajusten a las solicitadas en el entrenamiento con una tolerancia predefinida. Finalmente, cuando el error es menor al aceptado se genera un archivo con los pesos de la red neuronal que se llama `Network.finished`. Una vez concluido el entrenamiento la red neuronal usa el archivo de pesos cada que se le invoca.

Bibliografía

[1] Joseph L. Jones Anita M. Flynn. *Mobile robots, inspiration to implementation.* A K Peters, Massachusetts, USA, 1993.

[2] Rodney A. Brooks. New approaches to robotics. *Science*, september 1991.

[3] Patrick H. Winston. *Inteligencia Artificial.* Addison Wesley, 1992. Versión en español de la tercera edición en Inglés.

[4] Edward A. Feigenbaum. Knowledge engineering in the 1980's. Technical report, Depto. de Ciencias de la Computación, Universidad de Stanford,California, 1982.

[5] Joseph Giarratano Gary Riley. *Expert Systems, Principles and Programming.* PWS Publishing Company, Massachussets, USA, second edition, 1994.

[6] Joseph Giarratano. *CLIPS User's Guide.* Software Technology Branch, NASA, 1994.

[7] Simon Haykin. *Neural Networks: a comprehensive foundation.* Macmillan, 1994.

[8] Leonid B. Sheremetov. Estructuras de comunicación para la resolución de problemas de manera distribuida en la ingeniería concurrente. *Temas*, 1997.

[9] Jean-Claude Latombe. *Robot Motion Planning.* Kluwer Academic Publishers, Massachussets, USA, 1991.

[10] Tomás Lozano-Pérez Michael A. Wesley. An algorithm for planning collision-free paths among polyhedral obstacles. *Communications of the ACM*, 22(10):560–570, october 1979.

[11] Josep Tornero i Montserrat. Planificación y guiado de sistemas robotizados, master cad/cam. Material del Master CAD/CAM de la Universidad Politécnica de Valencia, España, 1997.

[12] N. J. McCormick. The mitchi road follower. conjunto de notas acerca de navegación y pilotaje, 1989.

[13] Vladimir J. Lumelski Alexander A. Stepanov. Dynamic path planning for a mobile automaton with limited information on the environment. *IEEE Transactions on Automatic Control*, 31:1058–1063,, 1986.

[14] B. H. Krogh D. Feng. Dynamic generation of subgoals for autonomous mobile robots using local feedback information. *IEEE Transactions on Automatic Control*, 34:483–493, 1989.

[15] Marco Colombeti Marco Dorigo Giuseppe Borghi. A methodology for behavior engineering. *IEEE Transactions on Systems, man and Cybernetics-part B: Cybernetics*, 26(3), june 1996. revista de la IEEE.

[16] Roger S. Pressman. *Ingeniería del software: Un enfoque práctico*. McGraw-Hill, third edition, 1993.

[17] David W. Rolston. *Principios de Inteligencia Artificial y Sistemas Expertos*. McGraw-Hill, 1992. Traducción al español de la 1a edición en inglés.

[18] John J. Leonard Hugh F. Durrant-White. *Directed sonar sensing form mobile robot navigation*. Kluwer Academic Publishers, 1992.

[19] Real World Interface, New Hampshire, USA. *BeeSoft User's Guide and Software Reference*, 1997.

[20] Russell R. Leighton. *The Aspirin/MIGRAINES Neural Network Software: User's Manual*. MITRE Corporation, USA, 1992.

[21] Jesús Savage-Carmona Marco Morales-Aguirre. Herramientas gráficas en clips para simulación de robots. ponencia presentada en el coloquio: La Investigación en la Facultad de Ingeniería, la versión de CLIPS modificada está disponible en ftp://odin.fi-b.unam.mx, 1997.

[22] Kay A. Robbins Steven Robbins. *UNIX, programación práctica: Guía para la concurrencia, la comunicación y los multihilos*. Prentice Hall, 1997. Traducción del inglés de: Practical UNIX programming: A guide to concurrency, communication and multithreading.