PROYECTO FINAL DE CARRERA

DESARROLLO DE UNA INTERFÁZ GRÁFICA PARA LA NAVEGACIÓN DE UN ROBOT MÓVIL

DELFÍN PEREIRO

2001

ÍNDICE DE CONTENIDOS

1 OBJETIVO, FINALIDAD Y OBJETO

1.1. OBJETIVO

El proyecto *Desarrollo de una interfaz gráfica para la navegación de un robot móvil*, tiene como objetivo la implementación de una interfaz gráfica de usuario que permita la comunicación e intercambio de datos con el robot MARCO.

La interfaz debe ser capaz de dotar al usuario de un control preciso y en tiempo real de las capacidades motoras del robot, haciendo posible el control directo mediante dispositivos de entrada (teclado, joystick y ratón). El control directo del robot debe incluir la posibilidad de alternar entre el control del desplazamiento del robot, y el control de la orientación de la cabeza.

MARCO dispone de la posibilidad de activar comportamientos que le permiten realizar tareas simples, como trasladarse a velocidad constante, evitar colisiones, etc... mediante la utilización de comandos. La interfaz debe permitir al usuario activar y desactivar estos comportamientos, además de modificar sus parámetros, sin necesidad de escribir comandos, gráficamente.

El usuario ha de disponer de la información de los sistemas de percepción activa, que incluyen sonares, cámaras e información odométrica. La información de los sonares debe ser representada junto una simulación del movimiento del robot real. De esta manera el usuario tiene una noción más exacta de la situación en que se encuentra el robot, así como de la información que utiliza en la toma de decisiones cuando se activan los comportamientos. Es importante disponer en la interfaz de una zona que muestre imágenes de lo que captan las cámaras para facilitar más aún el control del robot, y la orientación del usuario.

Por último la interfaz ha de ser capaz de generar trayectorias libres de obstáculos para MARCO, según un punto de destino definido por el usuario

1.2 FINALIDAD

Para la navegación en ambientes de características no constantes, un robot requiere de sensores que le permitan detectar la presencia de obstáculos para obrar en consecuencia. Toda la información obtenida a partir de estos sensores debe ser procesada y analizada antes de tomar las decisiones pertinentes. Un robot puede navegar de manera autónoma, tomando decisiones por si mismo en función de los estímulos externos, o de manera guiada, en cuyo caso es una persona la que toma las decisiones referentes al movimiento. Tanto en un caso como en el otro, es necesario comunicarse con el robot de manera que pueda obtenerse la información de sus sensores, tener acceso a sus funciones de toma de decisiones, modificar sus parámetros de conducta, etc... Todo esto se realiza a través de una interfaz gráfica de usuario.

En un robot como MARCO, dotado de varios tipos de sensores (ultrasonidos, parachoques, cámaras para la realización de estéreo-visión), se debe tener en cuenta gran número de variables, y los distintos tipos de operaciones disponibles precisan de la definición de gran número de parámetros. Las interficies de control de robots han evolucionada mucho en los últimos años, pero la manera más sencilla, eficiente y rápida siguen siendo las interfases gráficas.

1.3 OBJETO

En este proyecto se abordará el desarrollo de una interfaz gráfica que permita guiar el robot MARCO en ambientes interiores.

Lo que se busca es disponer de los elementos de ayuda necesarios, en la guía y seguimiento del robot móvil MARCO. En el proyecto se plantea el seguimiento del robot, la representación del mismo y de la información de los sensores, y el acceso y definición de los parámetros de configuración necesarios, tanto en la propia interfase como en los comportamientos del robot y path planning.

La comunicación con el robot puede dividirse en varias partes. La comunicación hombre-máquina la realiza la interfaz gráfica, con el fin de ser un intermediario intuitivo y eficiente en el control del robot por parte del usuario. También existe la comunicación máquina-máquina, que es la que se produce entre el centro de control y el robot (comunicación remota vía radio frecuencia), o entre robot (concretamente el ordenador de

a bordo del robot) y el Pan&Tilt, que puede catalogarse de comunicación local, ya que se produce en el robot.

El control del robot debe permitir un guiado manual durante una posible fase de aprendizaje del entorno, así como un control más autónomo del usuario.

1.4 CONTENIDO DEL PROYECTO

La información del proyecto esta estructurada en los siguientes capítulos:

- **Capítulo 1:** Objetivo, Finalidad y Objeto. Este capítulo.
- **Capitulo 2:** Planteamiento del diseño de la interfaz. En este capítulo se realiza una breve descripción de algunas interfases gráficas ya desarrolladas, para a continuación, enumerar los criterios utilizados en el diseño de la interfaz gráfica de este proyecto.
- **Capítulo 3:** Descripción de los componentes hardware y software del proyecto. Se describen los diferentes subsistemas que componen el entorno de trabajo de este proyecto, así como sus componentes, tanto Hardware como Software.
- **Capítulo 4:** Seguimiento y representación del robot y sus sensores. En este capítulo se tratan los diferentes procesos a seguir para realizar el seguimiento del robot, y la representación de la información de sus sensores, en la interfaz.
- **Capítulo 5:** Control de los sistemas mecánicos del robot. Aquí se enumeran los diferentes modos de control de los sistemas mecánicos (robot y Pan&Tilt) y sus problemas derivados, así como las soluciones adoptadas en el proyecto.
- **Capítulo 6:** Generación de trayectorias. En este capítulo se trata la problemática de generar una trayectoria libre de obstáculos para un robot móvil.
- **Capítulo 7:** Resultados. Se ofrecen resultados obtenidos con la interfaz desarrollada, en varios aspectos de control y representación.
- **Capítulo 8:** Conclusiones. En este capítulo se exponen las conclusiones extraídas del desarrollo de la interfaz, así como propuestas de mejora.

2 PLANTEAMIENTO DEL DISEÑO DE LA INTERFAZ

2.1. INTRODUCCIÓN

En este capítulo se hace mención a los pasos previos al desarrollo de la interfaz.

Una vez planteados los objetivos del proyecto, se hace necesario un estudio de los sistemas desarrollados para otros proyectos de índole similar, a fin de determinar las soluciones adoptadas ante los problemas que se plantean en un proyecto de estas características. Más adelante, en este mismo capítulo, se establecen los criterios que determinarán la apariencia de la interfaz de usuario que se pretende desarrollar, así como las funciones y utilidades que deberá presentar.

2.2. SISTEMAS DESARROLLADOS

Teniendo en cuenta los objetivos de este proyecto, se ha recopilado información a cerca de diferentes interfases de usuario o GUI's (Graphical User Interface), que puedan servir de referencia a la hora de elaborar un diseño apropiado.

La referencia más cercana de que se dispone, y por tanto la que se comenta más profundamente en este apartado, es la propia interfaz que provee Saphira para el control del robot Pioneer. En este caso la representación del mapa se realiza en 2D, y los objetos del mundo se modelan mediante conjuntos de rectas. El control más básico se realiza mediante los cursores del teclado y la barra espaciadora, mientras que otras tareas más complejas, como la carga de comportamientos y la variación de sus parámetros, puede realizarse mediante comandos o la carga de ficheros externos.

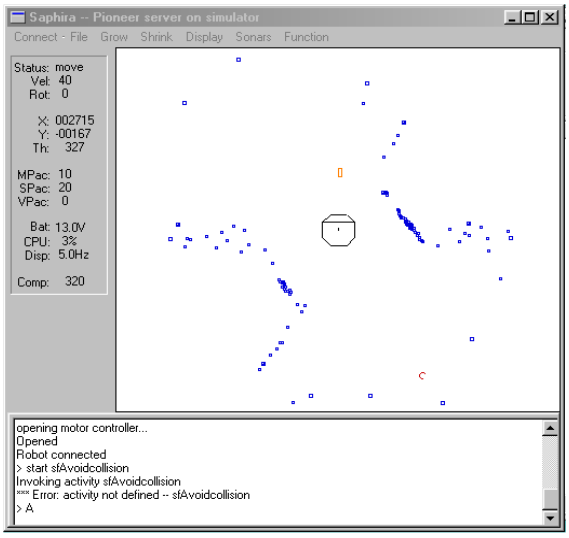


Figura 2.1: Saphira es un ejemplo clásico de interfaz gráfica 2D

La interfaz permite hacer un zoom sobre el mapa, así como la interesante posibilidad de representar la información centrada en el mundo, es decir, el mundo esta quieto y es el robot el que se desplaza sobre el mapa; o centrada en el robot, es decir, el robot quieto en el centro del plano, y es el mundo el que se mueve alrededor de él.

La plataforma de trabajo puede ser UNIX, o cualquier versión de Windows, ya que existen diferentes versiones de Saphira para cada sistema operativo.

Después de un breve estudio de la interfaz de Saphira se buscaron otras referencias con las que elaborar los criterios de diseño de la interfaz del proyecto. La gran mayoría de interfases utilizan un mapa 2D como medio de representar las evoluciones del robot. La simplicidad se impone ante otras soluciones más complejas como la utilizada en el proyecto AVENUE del laboratorio de robótica de la universidad de Columbia (ver *Figura2.2*)

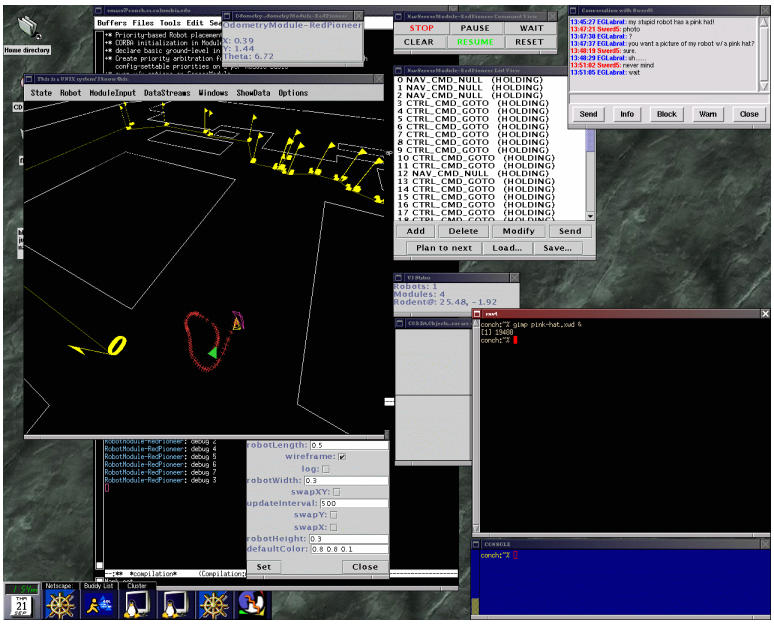


Figura 2.2: Interfaz de navegación del proyecto AVENUE

En general, la forma de la interfaz, así como la información que muestra y la disposición de esta, viene dada en gran medida por las características del robot que pretende controlar y los sensores de que esta dotado. Otros factores que intervienen en el desarrollo de las mismas son las capacidades motoras del robot, así como la aplicación del mismo. Un buen ejemplo de esto sería la interfase de la *Figura 2.3* (página siguiente).

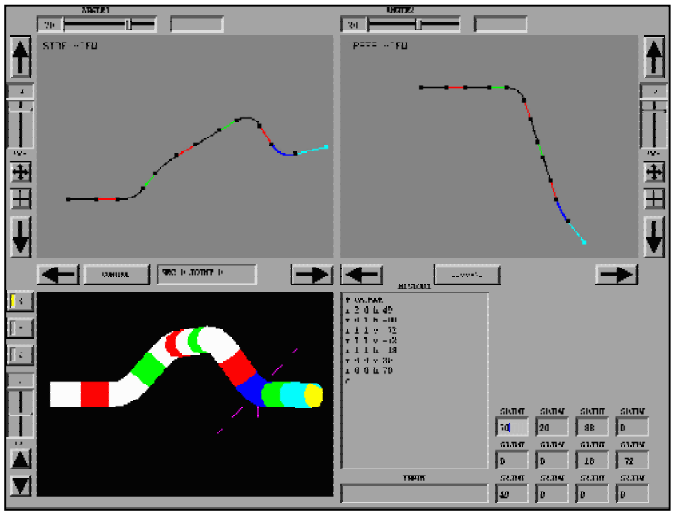


Figura2.3: Interfase de control de un robot tipo serpiente

En este caso se trata de la interfase de control de un robot que trata de simular el movimiento de un serpiente. Como es lógico, la interfase de control presenta un configuración totalmente diferente a la de otras interfases que controlan robots más convencionales, aunque como se puede apreciar, el objetivo de la interfaz es el mismo que en todas las demás, permitir al usuario el seguimiento y control de las acciones del robot.

La conclusión que se extrae de este breve repaso por algunas de las interfases de control que se han estudiado para este proyecto, es que no existe una manera estándar de implementar interfases gráficas. En cada caso debe realizarse un estudio de las capacidades del robot, así como de sus futuras aplicaciones, a fin que usuario pueda disponer de las herramientas necesarias para obtener información de sus sensores, y controlar eficazmente el robot.

Por tanto, los criterios que se utilizarán en diseño de la interfaz gráfica de este proyecto, estarán basados en las necesidades de control, y capacidades concretas del robot MARCO.

2.3. CRITERIOS UTILIZADOS EN EL DISEÑO DE LA INTERFAZ GRÁFICA

La interfaz debe ser amigable e intuitiva. Se busca la simplicidad de manejo, con la intención que la interfaz pueda ser usada como medio para controlar el robot, por alguien que no posea grandes conocimientos de informática. Con este objetivo la interfaz se desarrollará para el sistema operativo Windows (95, 98, 2000, ME o NT), ya que se trata del más extendido entre usuarios con conocimientos informáticos medios/bajos.

La comunicación y el control han de ser en tiempo real. A pesar de que la comunicación del robot con el centro de control se realiza mediante un radio Ethernet, el usuario debe ser capaz de controlar todas las funciones del robot, y recibir información de este, en tiempo real.

El robot MARCO posee la capacidad de “adoptar” ciertos comportamientos que le permiten ejecutar tareas algo más complejas que una simple translación, como mantener una velocidad constante, evitar obstáculos, y otras. La interfaz debe ser un medio flexible y potente de controlar la activación y desactivación de estos comportamientos, así como de modificar sus parámetros en tiempo real.

La generación de trayectorias libres de obstáculos debe ser simple e independiente, a fin que las intervenciones del usuario sean las mínimas necesarias en este proceso, y que los pasos a realizar por este , sean lo más simples posible.

3 DESCRIPCIÓN DEL HARDWARE Y SOFTWARE

3.1 INTRODUCCIÓN

Este capítulo trata de situar el problema describiendo los componentes hardware y software que componen el sistema.

Debido a la gran cantidad de dispositivos que conforman el sistema, se abordará su descripción dividiendo el problema en subsistemas más pequeños interrelacionados entre sí.

El sistema se puede dividir en los siguientes subsistemas:

1) Robot
2) Cabeza
3) Control de Navegación

3.2 ROBOT

El subsistema del robot esta compuesto por el Robot Pioneer 2DX con sus sensores asociados y un transmisor inalámbrico Ethernet que mantiene la conexión del robot con el control de navegación. La Conexión del robot con la "cabeza" se realiza a través del puerto serie (COM2) del ordenador de a bordo del robot.

3.2.1 DESCRIPCIÓN DEL HARDWARE

3.2.1.1. ROBOT PIONEER 2-DX

El robot Pioneer 2-DX tiene unas dimensiones de 44cm x 38cm x 22 cm, y su peso, con una batería, es de 9 kg, pudiendo soportar un peso máximo adicional de 23 Kg a bajas velocidades. Su autonomía de entre 18 y 24 horas con sus tres baterías plenamente cargadas.

Figura 3.1: Robot Pioneer 2-DX

Dotado de dos ruedas motrices, cada una de ellas viene equipada con un motor reversible de corriente continua, y una pequeña rueda trasera a modo de soporte, llamada en la literatura inglesa "caster". Estas ruedas motrices le permiten trasladarse a una velocidad máxima de 1,6 metros por segundo, así como rotar sobre si mismo haciendo girar ambas ruedas en sentido opuesto, o girar sobre una de ellas estacionaria, con un radio de giro de 32 cm.

Los sensores con los que cuenta el robot son:

- 16 sonares distribuidos alrededor del robot.
- Parachoques o "Bumpers"
- Contadores de posición o "Encoders"

Sonares

El sonar es un sensor que permite detectar la distancia aproximada a la que se encuentra un obstáculo. El sistema se basa en la emisión de un pulso sonoro que, en caso de rebotar contra un objeto, es recogido por el mismo sonar, el cual estima la distancia a la que se encuentra el objeto mediante la siguiente formula:

$$D = \frac{V \cdot T}{2} \qquad (3.1)$$

Donde:

D = Distancia del sonar al objeto
V= Velocidad del impulso sonoro (340 m/s)
T= Tiempo transcurrido desde la emisión del pulso hasta que vuelve a ser recibido por el sonar

Los principales inconvenientes de estos sensores son:

- La emisión del sonar no es lineal, por lo que no se puede calcular la posición exacta donde se ha producido la lectura, solo la zona aproximada (ver *Figura 3.2*).

- Si la superficie sobre la que se refleja el pulso sonoro forma 30º o menos con la vertical del sonar, el reflejo se "pierde" y no es captado por el sonar (ver *Figura 3.3*).

- Si el pulso sonoro rebota varias veces antes de ser captado por el sonar, este dará una lectura falsa. El tiempo transcurrido desde su emisión hasta que es recibido no corresponde con una distancia lineal, y por tanto pensará que el obstáculo se encuentra mucho más lejos de lo que en realidad está (ver *Figura 3.4*).

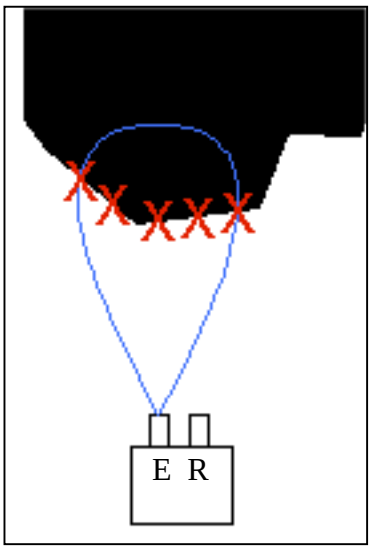


Figura 3.2: Cualquiera de los puntos del contorno del objeto que quedan abarcados por el lóbulo, pueden dar una lectura positiva

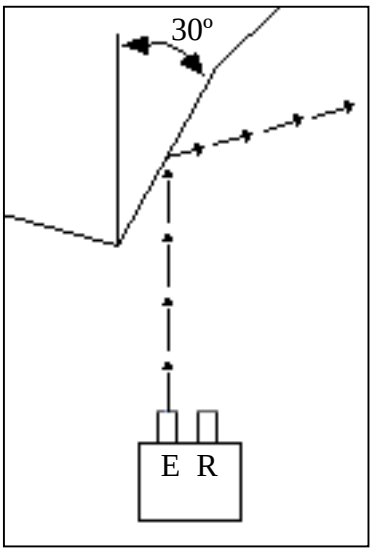


Figura 3.3: Como se puede observar, el impulso sonoro rebotaría sin llegar al receptor, perdiéndose.

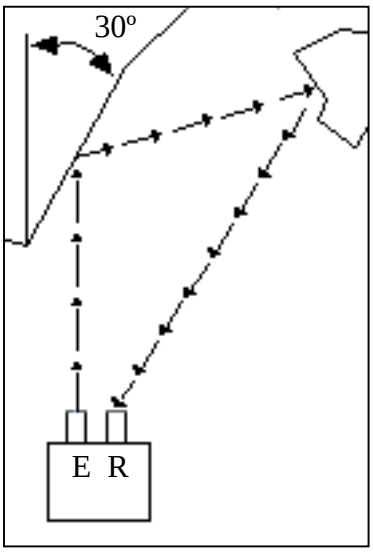


Figura 3.4: El impulso sonoro llega al receptor después de 2 o más rebotes con lo que se da una lectura a una distancia errónea.

El robot tienen dos anillos de 8 sonares cada uno, uno delantero y uno trasero. La disposición de estos, como se aprecia en la *Figura 3.5* (ver página siguiente), sitúa un sonar a cada lado del robot y los seis restantes a intervalos regulares de 20º.

Mientras que los sonares frontales y traseros, posibilitan la navegación del robot detectando posibles obstáculos; los sonares laterales (dos del anillo frontal y dos del trasero), además de colaborar en la detección de obstáculos durante la navegación (sobretodo en lo que a giros se refiere), facilitan el seguimiento de paredes, corredores, etc

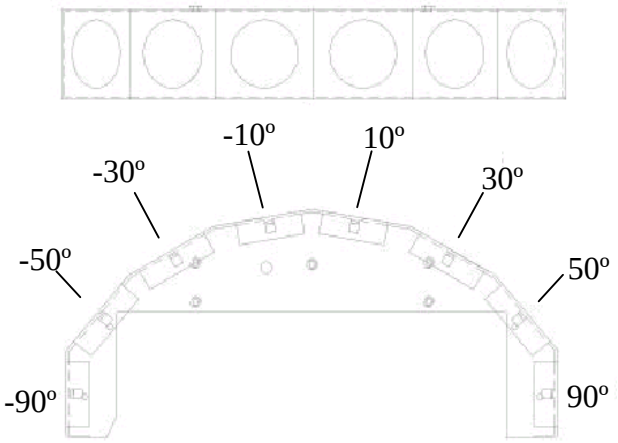


Figura 3.5: Distribución de un anillos de sonares del Pioneer 2-DX . Tanto el anillo frontal, como el trasero son idénticos.

La frecuencia de disparo de los sonares es de 25 Hz (40 milisegundos) y el rango de sensibilidad va desde los 10 cm a 5 metros o más.

Contadores de posición o "encoders"

Los "encoders" se encargan de obtener la información odométrica del robot. La fracción de vuelta de cada rueda motora junto al radio de estas, permite saber el desplazamiento realizado por cada una de ellas. En este caso concreto, el robot viene equipado con un "encoder" en cada motor que proporciona una resolución de 9,850 lecturas por cada vuelta de rueda, que equivale a una precisión de 19 lecturas por milímetro.

La información odométrica del robot permite conocer la posición aproximada del robot en cada momento. El inconveniente de estos sensores es que, aunque como en este el caso, los "encoders" en sí dan una resolución superior a la décima de milímetro, la información no es lo suficientemente fiable. Debido a que el robot puede realizar sus evoluciones sobre terrenos muy dispares (moqueta, parquet, etc ...), con propiedades de deslizamiento y adherencia muy diferentes entre si, alguna de las ruedas motoras puede patinar sobre el terreno, provocando un pequeño error en la estimación de la posición del robot. Este error, aunque insignificante en principio, debido a la suma de errores acumulados, puede llegar a provocar grandes errores en la estimación de la posición. Este problema se conoce como "dead reckoning", y su mayor inconveniente es que pequeñas variaciones en la orientación del robot se traducen en grandes variaciones en la traslación.

Supóngase el caso que el robot ha realizado un giro determinado, y se produce un error de 1 grado en la estimación de dicho giro. Si a continuación el robot se desplaza 1 metro en la dirección en la que esta orientado, se obtiene un error en la estimación de la posición de 17 milímetros aproximadamente, y así sucesivamente.

Bumpers

Situados a modo de "parachoques", los "bumpers" son unas pequeñas placas metálicas distribuidas alrededor del robot, cuyo objetivo es, en último termino, la detección de la colisión del robot con algún objeto del entorno. La única información que se puede obtener de ellos es si hay contacto o no.

3.2.1.2. RADIO ETHERNET

El AirEZY2400 es un dispositivo de conexión de red LAN (red local) vía radio. La frecuencia de emisión del dispositivo es de 2,442 GHz, con una cobertura de 23.225 metros cuadrados. La alimentación la suministra un pequeño transformador que da 5 voltios en continua (DC) y 2000 miliamperios.

3.2.2 DESCRIPCIÓN DEL SOFTWARE

3.2.2.1 SAPHIRA

Saphira es un entorno de desarrollo de aplicaciones independiente de la plataforma software, escrito y en constante actualización por SRI International's Artificial Intelligence Center bajo la dirección del Dr. Kurt Konolige.

El software nativo del Pioneer 2-DX es Saphira. Su sistema de trabajo en entorno cliente/servidor permite, mediante el conjunto de rutinas incluidas en las librerías Saphira, desarrollar programas-cliente capaces de comunicarse con el robot, así como tener acceso y control de sus sensores y motores.

Colbert

Saphira también incorpora un lenguaje de programación para robots móviles, muy similar a C, denominado Colbert. Gracias a este lenguaje el usuario puede escribir rápidamente complejos procedimientos de control llamados actividades.

Como ya se ha mencionado anteriormente, Colbert es muy similar a C en su sintaxis pero posee las siguientes limitaciones:

- Los tipos básicos son `int`, `float`, `void` y `string`. No existen los tipos `double` o `char`.
- No se pueden definir estructuras. Estas deben ser importadas de un fichero objeto en C nativo.

- No existen las listas ni los operadores de listas.
- Los siguientes operadores no existen: ?:, op=, >> ,<<, ++, --.
- El casting explicito de variables no esta permitido (aunque se realiza un casting implícito).
- Las declaraciones `for` y `switch` no están definidas.
- No se pueden inicializar la variables al mismo tiempo que se definen.
- No se permiten las asignaciones insertadas, como por ejemplo: `if ((x = a ) > 2) {...}`.
- Nuevas funciones no pueden ser definidas en Colbert, pero pueden ser importadas de un fichero objeto en C nativo.

Compilador de comportamientos

Saphira usa reglas de control difuso para implementar programas de control de bajo nivel, o comportamiento ("behaviors" en la literatura inglesa). Estos "behaviors" son definidos utilizando estructuras y funciones estándar de C.

Para hacer más sencilla al usuario la tarea de programar estos comportamientos, Saphira dispone de un Compilador de comportamientos que traduce unas simples reglas de control difuso, en el código en C requerido para implementarlas. Los comportamientos, siendo un tipo de actividad, pueden ser activados y desactivados al igual que estas.

Simulador del robot

Aunque no hay nada como las condiciones del mundo real para comprobar el buen funcionamiento de nuestras implementaciones, Saphira incorpora un simulador por software del robot y su entorno. Este simulador es una alternativa muy útil a su homónimo real, ya que nos permite comprobar el resultado de nuestros programas, comportamientos, etc ... sin el riesgo que esto podría significar para el robot.

Dentro de las capacidades del simulador están las de reproducir el funcionamiento de los sonares y los "encoders" con modelos realistas de errores; es decir, tanto las lecturas de los sonares como los datos de odometría, simulan los mismos errores que se producen durante la adquisición de datos con el robot real (errores anteriormente mencionados en el apartado 3.2.2.1).

Como se puede observar en la *Figura3.10*, el simulador también permite proporcionarle un mundo representado por segmentos, por el cual el robot virtual puede navegar. Es aquí donde el simulador nunca podrá sustituir las pruebas en un entorno real, ya que el mundo virtual creado para el simulador es una abstracción 2D. Esto significa que el simulador asume que todo lo que aparece representado en el mundo virtual, puede ser detectado por los sonares.

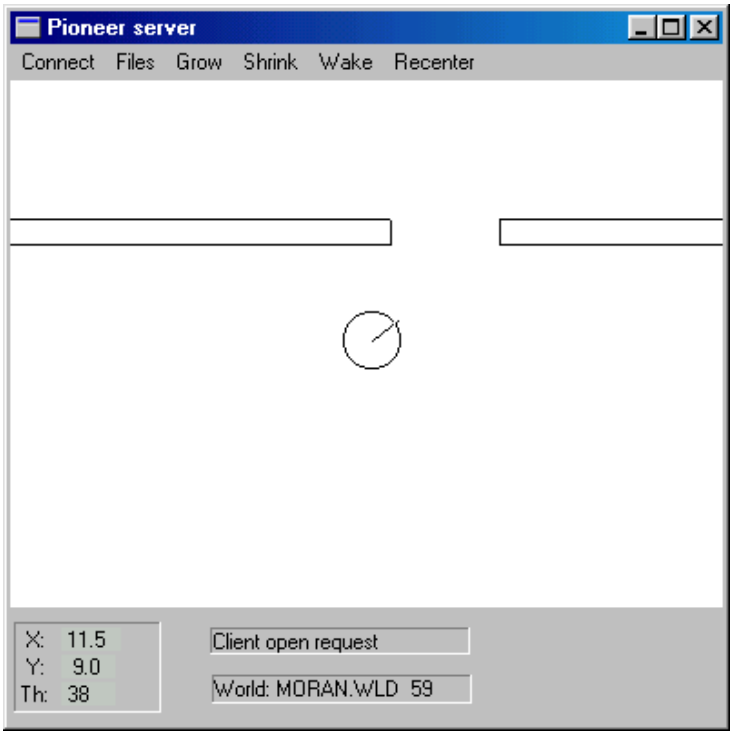


Figura 3.10: Imagen del simulador por software incluido con Saphira

En este proyecto el robot dispone de una cabeza equipada con dos cámaras para realizar estéreo-visión. Si durante la navegación el robot se encuentra con una mesa, los sonares solo detectarán sus patas pero no la parte superior de la mesa, lo cual puede poner el peligro la integridad de la cabeza. Este es el tipo de situaciones que no pueden ser planteadas en el simulador.

3.2.2.2 IPTHRU

IpThru es un programa desarrollado con el objetivo de recibir los paquetes de información que envía un cliente Saphira, a través de una LAN (Red Local). Una vez se establece la conexión con el cliente, IpThru redirecciona las instrucciones al puerto serie (COM1) del ordenador de a bordo del Robot, donde se encuentra conectado el microcontrolador del mismo.

3.3 CABEZA

El sistema definido como cabeza, se compone de Pan&Tilt, cámaras y video-senders. Como ya se mencionó en el apartado 3.2, la conexión de la cabeza con el robot se realiza a través del puerto serie (COM2) del ordenador de a bordo del robot, mientras que la información de las cámaras se envía al centro de control y navegación a través de los video-senders.

3.3.1 DESCRIPCIÓN DEL HARDWARE

3.3.1.1 PAN&TILT

El Pan&Tilt es un elemento mecánico que permite la orientación de objetos en el espacio, en este caso las cámaras de estéreo-visión. Mediante dos motores se controla el valor de los ángulos de Pan (Giro) y Tilt (Cabeceo), dentro de unos límites físicos de ±139,8º de giro y +31,54º y –47,33º de cabeceo, entendiendo como giros positivos los representados en la *Figura3.6*.

La transmisión de datos se realiza vía puerto serie (COM2) del PC, a través del cual las ordenes le son enviadas al controlador del Pan&Tilt, que se encarga de interpretarlas.

El Pan & Tilt tiene dos modos de operación:

- Independent control mode: con este modo de control se establece el ángulo (absoluto o relativo) en que se pretenden orientar los ejes.
- Pure velocity control mode: En este modo de operación lo que se le comunica al Pan&Tilt no es un ángulo, sino la velocidad a la que se desea que giren los ejes.

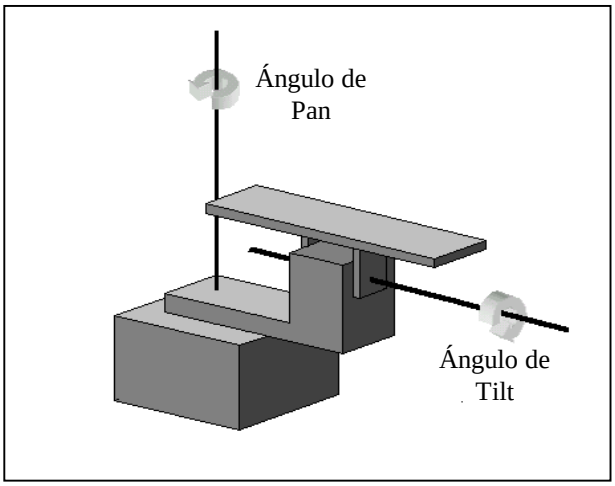


Figura 3.6: Esquema del Pan&Tilt con sus ejes de giro en sentido positivo.

En el apartado 5.2.3. se analizará más detalladamente estos modos de operación, así como las técnicas de control del Pan&Tilt más apropiadas para la interfaz de navegación.

3.3.1.2 CÁMARAS

Entre las capacidades del robot destaca la de obtener información del entorno mediante un proceso de estéreo-visión. A fin de obtener la información necesaria para tal proceso, el robot dispone de un Pan&Tilt (ver apartado 3.3.1.1.) sobre el que se han montado dos cámaras.

Las cámaras utilizadas son marca Sony, modelo EVI-371DG en color. El sensor que incorporan estas cámaras en un CCD de 1/3 pulgadas, con un ratio 4:3 entre la longitud horizontal y la vertical. La salida de datos se efectúa a través de una conexión de video compuesto PAL estándar.

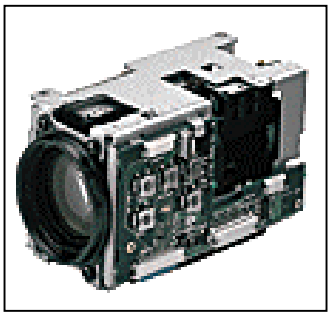


Figura 3.9: Imagen de la cámara SONY EVI-371DG

Para la obtención de un mapa de profundidades mediante la técnica de estéreo-visión, es muy importante que las imágenes de ambas cámaras sean obtenidas al mismo tiempo. A tal efecto se optó por este modelo, que permite una señal de sincronía externa proveniente de una señal de video de otra cámara.

3.3.1.3 VIDEO-SENDERS

Se trata de un dispositivo de emisión de video, distribuido por NSI Nevada Systems. La información se emite en formato de video compuesto (PAL), a una frecuencia de 900 MHz, con una covertura aproximada de 500 metros cuadrados. La alimentación es de 12~15 Voltios y 300 miliamperios.

3.3.2 DESCRIPCIÓN SOFTWARE

3.3.2.1. PAN&TILT CONECTION LISTENER (PTCL)

Con un funcionamiento y objetivo muy similares a los del IpThru (apartado 3.2.2.2.), pero dedicado al Pan&Tilt, este programa determina cuando llega información para el Pan&Tilt vía LAN (red local) y la redirecciona al puerto serie COM2.

3.4 CONTROL DE NAVEGACIÓN

Este sistema es el encargado de coordinar toda la información que se recibe tanto del robot, como del Pan&Titlt y las cámaras, así como de ejecutar las ordenes del usuario.

En este caso el control de navegación esta compuesto por un PC modelo Pentium III 600 MHz con 128 MB de RAM, una tarjeta de adquisición y procesado Matrox METEOR, un dispositivo Radio Ethernet (mirar sección 3.2.1.2) y un receptor de señales de Video (mirar sección 3.3.1.3.).

3.4.1 DESCRIPCIÓN DEL HARDWARE

3.4.1.1. TARJETA DE ADQUISICIÓN MATROX METEOR

La Matrox METEOR es una tarjeta gráfica de adquisición de imágenes. No dispone de memoria gráfica propia, por lo que las imágenes grabadas se almacenan directamente en la memoria del sistema (PC), o en la de la tarjeta VGA del sistema, dependiendo de la situación. Las librerías MIL (Matrox Imaging Library, ver apartado 3.4.2.1) utilizan uno de estos dos métodos para transferir los datos de la placa Matrox METEOR, a la memoria de visualización:

- Transferencia directa al buffer de muestreo (también llamada adquisición en vivo)
- Transferencia indirecta al buffer de muestreo (también llamada adquisición pseudo-vivo)

Transferencia directa al buffer de muestreo

La tarjeta de Meteor, generalmente puede enviar toda la información adquirida con una profundidad de color de 8 (monocromo), 15 y 32 bits (número de bits necesarios para codificar cada píxel) directamente a la memoria de muestreo, cuando la tarjeta VGA del sistema soporta accesos rápidos a memoria lineal (ver *Figura 3.7*).

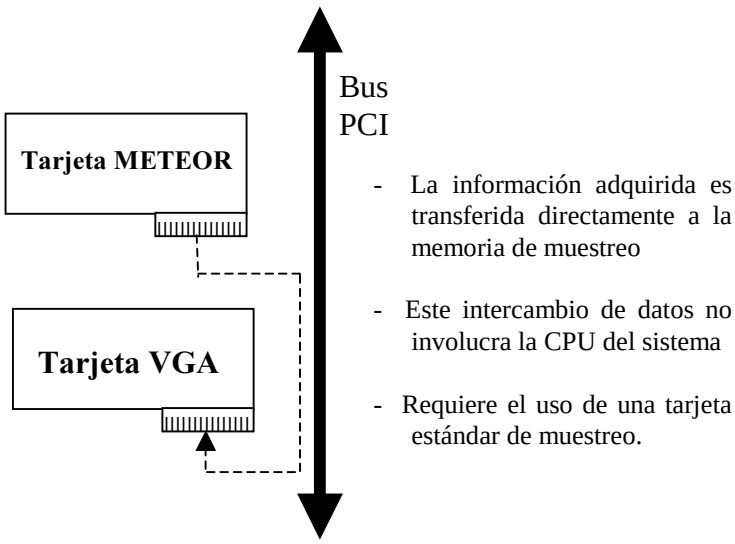


Figura 3.7: Transferencia directa de datos de METEOR a VGA

Transferencia indirecta al buffer de muestreo

Cuando la tarjeta VGA del sistema no soporta accesos rápidos a memoria lineal, la ventana en que se muestra la información está ocluida, o simplemente cuando el modo de muestreo no corresponde con el formato de grabación de datos de la placa Matrox Meteor (8, 15 o 32 bits por píxel), la transferencia directa de datos de la tarjeta Meteor a la tarjeta VGA del sistema no puede ser llevada a cabo. En estos casos, MIL realizará la transferencia automáticamente en modo indirecto.

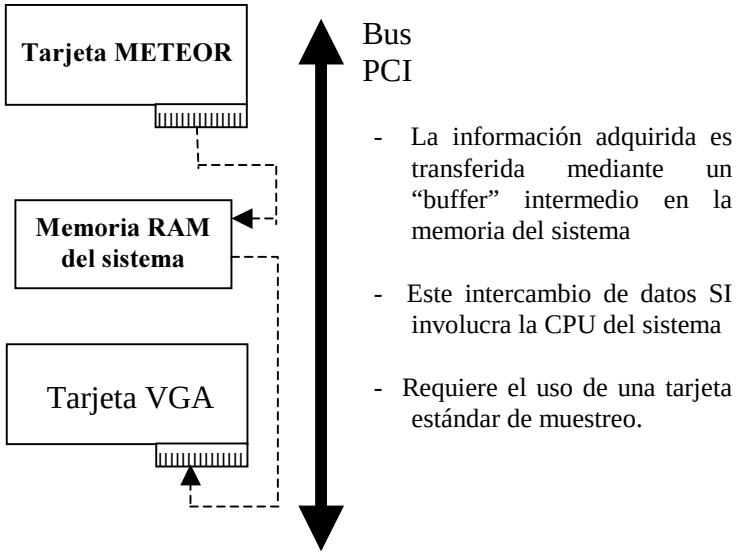


Figura 3.8: Transferencia indirecta de datos de METEOR a VGA

Cuando la trasferencia de datos es indirecta (o pseudo-vivo), la representación de la información será en tiempo real dependiendo de si la CPU es lo bastante rápida enviando información desde la memoria RAM del sistema a la tarjeta VGA. Esto quiere decir que la adquisición de imágenes en tiempo real puede verse afectada por la carga de trabajo que tenga la CPU, la cantidad de procesos en funcionamiento, sus prioridades, etc...

3.4.1.2. ORDENADOR DE NAVEGACIÓN

Como ya se mencionaba en la introducción de este apartado, el ordenador de navegación es un Intel Pentium III 600 MHz, con 128 Mb de RAM y un disco duro de 10 Gb, cuyo sistema operativo es Windows NT Server

3.4.1.3. RADIO ETHERNET

El AirEZY2400 es un dispositivo de conexión de red sin cable, vía radio. La frecuencia de emisión del dispositivo es de 2,442 GHz, con una obertura de 23.225 metros cuadrados. La alimentación la suministra un pequeño transformador que da 5 voltios en continua (DC) y 2000 miliamperios.

3.4.1.4. RECEPTOR DE SEÑALES DE VIDEO SENDER

Son los receptores de la señal de los video senders instalados en la cabeza del robot (ver sección 3.3.1.3). Se trata de unos dispositivos que, conectados directamente a una tarjeta de adquisición y procesado de imágenes (ver apartado 3.4.1.1.) permiten recibir la señal de video emitida por los video senders,

3.4.2 DESCRIPCIÓN DEL SOFTWARE

3.4.2.1. LIBRERÍAS MIL

Las MIL (Matrox Imaging Library) son un conjunto de librerías gráficas desarrolladas por Matrox para el tratamiento, análisis, creación y estudio de imágenes, tanto en color como en niveles de gris.

Estas librerías han sido desarrolladas con el fin de facilitar la implementación de aplicaciones gráficas bajo diferentes sistemas operativos (DOS, Windows 9X, Windows NT).

Aunque las funciones que contienen pueden ser ejecutadas sobre cualquier sistema (un PC con una tarjeta gráfica VGA estándar es suficiente), realizando en este caso todos los cálculos necesarios la CPU del ordenador, es sobre algunas de las tarjetas gráficas Matrox, tales como la Meteor, Pulsar, Genesis, etc... diseñadas para la captura, edición y proceso de imágenes, sobre las que se alcanza un mayor grado de eficiencia. No obstante, cabe mencionar que, para extraerle todo el rendimiento a estas tarjetas, es necesario recurrir a funciones específicamente diseñadas para ellas, como por ejemplo es el caso de las funciones contenidas en las librerías "Native" de Genesis, las cuales permiten un mayor control y aprovechamiento de las capacidades de esta.

A continuación se comenta brevemente los pasos necesarios para la implementación de una aplicación gráfica basada en las MIL:

1. En primer lugar se reserva memoria para la aplicación con la instrucción `MappAlloc()`. Esto crea el entorno de control y ejecución necesarios para la aplicación.

2. A continuación se reserva memoria para el sistema sobre el que hacemos funcionar la aplicación mediante la instrucción `MsysAlloc()`.
   El sistema representa el tipo de tarjeta gráfica sobre el que se esta ejecutando la aplicación, por tanto lo que se hace es abrir los canales de comunicación e inicializar los recursos del hardware.
   Tarjeta gráfica no tiene porque haber solo una en el sistema. Podemos tener una placa METEOR además de la tarjeta VGA del PC (como es nuestro caso), y puede convenirnos para nuestra aplicación reservar memoria para ambos sistemas (ver apartado 3.5., Conexión de la cámara de navegación).

3. A partir de este momento, con `MbufAlloc()`, se reserva el espació de memoria necesario, conocido como "buffer", para albergar la imagen con la que se pretende trabajar. Se deberá reservar memoria para un buffer por cada imagen que se quiera almacenar en memoria.

4. Si se pretende mostrar la imagen, así como las operaciones realizadas sobre ella, se ha de reservar memoria para la ventana sobre la que se realizará el muestreo con `MdispAlloc()`. No es imprescindible reservar memoria para un "display", solo si se desea mostrar la imagen.

5. Todas las operaciones necesarias se realizarán en los "buffers".

6. Finalmente, antes de finalizar la aplicación que hace uso de las librerías MIL, debe liberarse todo el espacio de memoria reservado empezando por el final, es decir, empezando por todos los "displays" con `MdispFree()`, a continuación todos los "buffers" con `MbufFree()`, el sistema con `MsysFree()`, y finalmente la aplicación con `MappFree()`.

4 SEGUIMIENTO DEL ROBOT EN UN MAPA BIDIMENSIONAL Y REPRESENTACIÓN DE LOS SISTEMAS DE PERCEPCIÓN ACTIVA

4.1. INTRODUCCIÓN

En este capítulo se tratan los sistemas empleados para el seguimiento y representación de las evoluciones del robot en un entorno real, así como los métodos utilizados para representar la información que nos llega de los sensores del robot (sonares, odometría, etc ...)

4.2. CARGANDO UN MAPA

El navegador dispone de una amplia zona sobre la que se representan las evoluciones del robot.

El seguimiento se puede realizar sobre 2 tipos de mapa:

1. Un mapa en blanco.
2. Un plano que modeliza una zona real.

El mapa en blanco es necesario para que, en un futuro, el robot pueda realizar sus propios mapas a partir de la información que le suministren los sensores. Algo así como el tapiz en blanco de un pintor.

En el estado actual del proyecto el robot aun no esta preparado para confeccionar sus propios planos. Por este motivo, el usuario tiene la opción de introducir un plano "hecho a mano" que le permita al robot hacerse una idea de donde esta.

4.2.1. DEFINICIÓN DE LAS ESTRUCTURAS DE DATOS NECESARIAS PARA EL MANEJO DE MAPAS

El mapa del mundo, predefinido por el usuario, se almacena en un fichero que describe las características del entorno en que el robot realizará sus evoluciones (ver apartado 4.2.2. Estructuración de archivos). Sin embargo, cada vez que se actualiza la vista del mundo en el Navegador, debemos de repintar el mapa, con lo que el constante acceso a disco penalizaría enormemente el rendimiento de la aplicación. A causa de esto, los datos del mapa actual deben de ser almacenados en memoria para un rápido acceso a ellos, así como correctamente estructurados para maximizar la eficiencia del algoritmo encargado del refresco del mapa, además de minimizar el espacio de memoria ocupado.

Así, la variable encargada de almacenar en memoria los datos del fichero de mundo es una lista de objetos, cada uno de los cuales tiene la siguiente estructura:

```
typedef struct {
   char *nombre;      // Nombre del objeto
   char *codigo;      // Código del objeto
   Pt2D Posicion;     // Posición que ocupa el objeto dentro el mundo
   double Orientacion;  // Orientación del objeto en el mundo
   nodo_ptr primero;    // Apuntador al primer punto del contorno del objeto
}poligono;
```

Figura 4.1: Estructura en C de los objetos que conforman un mapa

Es muy importante que el espacio de memoria reservado por cada uno de los campos de esta estructura sea el mínimo necesario, ya que en caso contrario, debido a la gran cantidad de objetos que pueden llegar a definir un mundo, un alto porcentaje de la memoria del sistema sería desperdiciada.

Por este motivo, tanto para el nombre del objeto como para el código de su estructura se reserva espacio en memoria dinámicamente, es decir, no hay una cantidad fija de memoria reservada para ellos, sino que se reserva la memoria justa para contener el número de caracteres que los forman. De esta manera es posible definir el nombre del

objeto y su código tan largo como se desee, el único límite es la memoria física del sistema, teniendo por seguro que jamás se ocupará indebidamente memoria que no va a ser utilizada.

Este hecho, se hace particularmente importante a la hora de reservar espacio en memoria para los puntos que definen el contorno del objeto. ¿Para cuántos puntos se debe reservar memoria?,¿10 puntos serían suficientes?, ¿menos?, ¿quizás más?; no es posible llegar a una solución totalmente satisfactoria. Si se reserva espacio para muchos puntos en la mayoría de los casos se reservaría memoria de más, malgastándola tal y como se ha comentado anteriormente, mientras que si se procura ajustar al mínimo el número de puntos de definen el contorno de los objetos, el usuario siempre estaría limitado. Una solución a este problema es el uso de listas encadenadas.

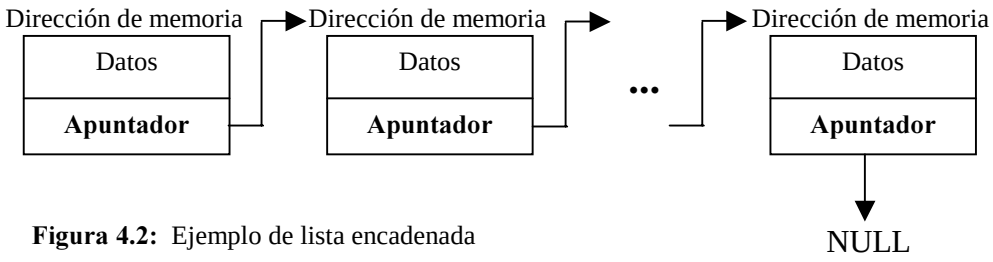


Figura 4.2: Ejemplo de lista encadenada

Las listas encadenadas se basan en la idea que, si se ha de almacenar en memoria la misma estructura de datos reiteradamente, es posible almacenar junto a dicha estructura un apuntador a una dirección de memoria que contenga otra estructura idéntica (naturalmente con valores diferentes, pero con la misma estructura de datos). Como se puede ver en la *Figura4.2.* , uno de los campos de la estructura de datos almacena la dirección de memoria de la siguiente estructura. Una estructura apunta a la siguiente y así sucesivamente, como si de eslabones de una cadena se tratara, hasta que el último eslabón apunta a NULL, como señal del final de la cadena.

Esta estructuración de la información permite añadir un nuevo paquete de datos en cualquier punto de la cadena de la manera más sencilla posible. Si, por ejemplo, se pretende añadir un nuevo paquete de datos en la posición **N** de la cadena, deben seguirse los pasos siguientes:

- Reservar memoria para el nuevo paquete de datos
- Especificar que el apuntador del eslabón **N-1** apunte hacia la dirección de memoria del paquete que se pretede añadir
- El apuntador del nuevo eslabón debe apuntar hacia la dirección de memoria del que antes era el eslabón **N**, que pasa a ser el **N+1**.

La *Figura 4.3.* (página siguiente) muestra más claramente este proceso.

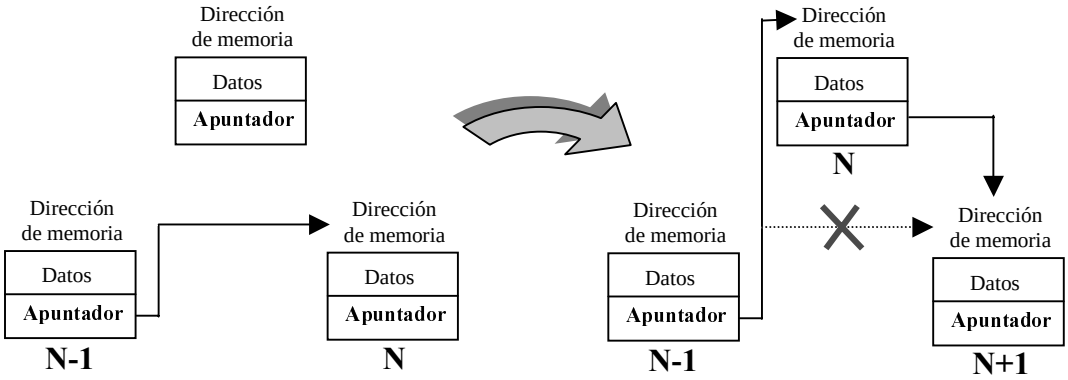


Figura 4.3: Ejemplo de inserción de datos en una listan encadenada

En este caso concreto, la información que se pretende almacenar son los puntos 2D que definen el contorno de los objetos, por tanto, la estructura que tendrá cada eslabón de la cadena es la siguiente:

```
typedef struct nodo{
   Pt2D value;              //Punto del contorno del objeto
   struct nodo *next;  //Apuntador al siguiente punto del contorno del objeto
}nodo;
```

Figura 4.4: Estructura en C de los puntos que definen el contorno de los objetos

Definiendo el apuntador a la estructura nodo, como nodo_ptr (*typedef nodo *nodo_ptr;* en C), se obtiene la definición del campo que se encarga de apuntar al primer nodo del contorno del objeto, dentro de la estructura que define los objetos del mapa (véase de nuevo la *Figura 4.1.*).

Ahora que la estructura de datos se encuentra completamente definida, hay que establecer la sintaxis de los archivos de datos.

4.2.2. ESTRUCTURA DE LOS ARCHIVOS

Como ya se mencionó en el capítulo anterior, la información de los mapas predefinidos por el usuario se almacenan en unos ficheros con extensión “map”. En realidad, estos archivos son unos simples ficheros de texto que llaman a otros, los ficheros de objetos, donde se encuentra la información específica de cada objeto. El fichero de mundo simplemente llama al objeto, lo posiciona y lo orienta.

El motivo por el cual se ha separado la descripción de los objetos, de la descripción del mundo en sí, es que esta distribución de los datos permite invocar el mismo objeto tantas veces como se desee (tanto dentro del mismo mapa, como en otros), sin tener que repetir la descripción del objeto explícitamente cada una de las veces.

A continuación, se describe la estructura de estos ficheros, tanto los ficheros de mundo, que describen el entorno en general; como los ficheros de objeto, que se encargan de determinar el contorno de los objetos, así como su nombre y codificación.

4.2.2.1. ESTRUCTURA DEL FICHERO DE MUNDO

Los ficheros de mundo contienen las referencias a los objetos que deben aparecer en el mapa, así como la posición y orientación de estos. También determinan otros aspectos importantes como la posición y orientación que ocupará el robot dentro del mapa, en el momento de la conexión.

En la *Figura 4.5.* tenemos un ejemplo de fichero de mundo. Como se puede observar, después del número de objetos que tiene el mapa, aparece el nombre del fichero que contiene la información del contorno. Por regla general, la llamada a los distintos objetos que componen el mapa se realiza especificando el nombre del fichero que contiene su descripción, y a continuación, la posición que ocupa el objeto (en coordenadas absolutas), y orientación.

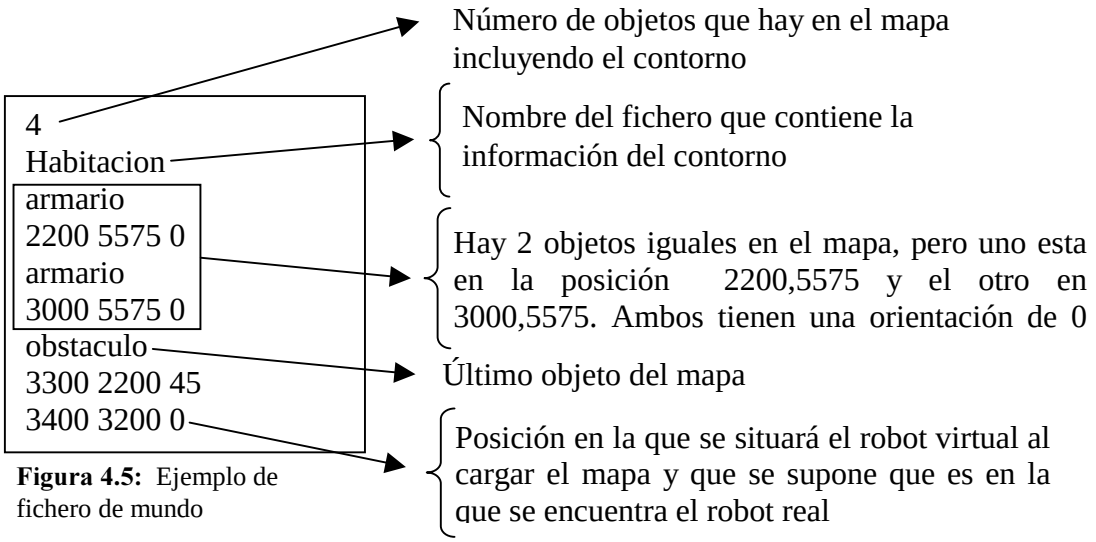


Figura 4.5: Ejemplo de fichero de mundo

Es importante no especificar la extensión de los ficheros que contienen la información de los objetos, ya que el programa la presupone. Si por ejemplo se escribe en el fichero de mundo “armario.dat”, el programa buscaría un objeto en el directorio \Objetos\ de nombre “armario.dat.dat” (que lógicamente no encontraría),

El único objeto que no sigue esta norma es precisamente el contorno del mapa que, a diferencia del resto de los objetos, no precisa ser posicionado ni orientado. El porqué de esta peculiaridad encontrará su explicación en el apartado siguiente (4.2.2.2. Estructura del fichero objeto), donde se ahondará más en el tema.

Finalmente, una vez descritos todos los objetos con sus respectivos datos de posición y orientación (a excepción del contorno, tal y como ya se ha comentado), solo queda establecer la posición y orientación que ocupará el robot virtual dentro del plano. Esta característica es muy útil, ya que el robot real no siempre estará igualmente posicionado y orientado dentro del mundo real

4.2.2.2. ESTRUCTURA DEL FICHERO OBJETO

Los ficheros de mundo contienen la información específica del objeto que representan, así como un nombre y código que los identifican.

La *Figura4.6.* muestra un típico ejemplo de fichero objeto, donde en este caso se define un armario de 80x45 cm.

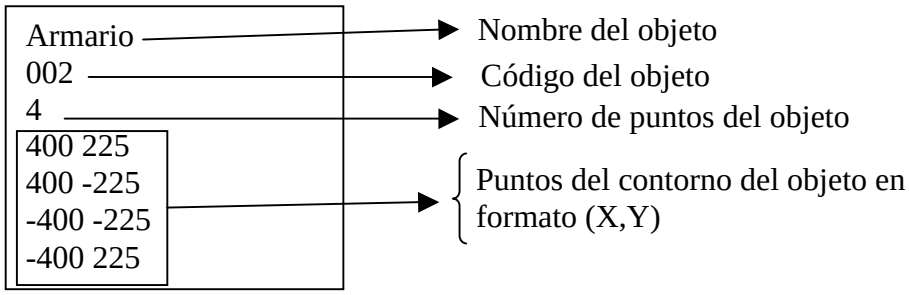


Figura 4.6: Ejemplo de fichero de objeto

El nombre del objeto designa el nombre con el que se pretende que el programa reconozca internamente el objeto. Por el momento, no tiene más implicaciones que la meramente estética, ya que el Navegador no reconoce los objetos ni realiza ningún tipo de operaciones con ellos. En un futuro, cuando sea necesario acceder a los datos de un objeto del entorno, su identificación se efectuará mediante el nombre.

El código con el que se designa el objeto tiene una relevancia más inmediata que la del nombre. Lo que este código representa es, de alguna manera, la movilidad del objeto dentro del mundo. Su significado se corresponde con la siguiente tabla

001: Contorno del mundo
002: Objeto del mundo que "siempre" ocupa la misma posición. Aquí podríamos suponer un armario, una mesa, etc ...
003: Objeto móvil del mundo. Podríamos suponer una persona, una perro, una silla nueva que han traído al despacho y que hasta ahora no había sido incluida en el mapa, etc...

El porqué de una codificación de 3 dígitos cuando solo hay 3 tipos de objetos, viene dado por una necesidad futura de ampliar la clasificación de objetos.

Una vez establecidos el nombre y la codificación del objeto, se procede a la descripción del mismo mediante los puntos de su contorno. Una vez declarada la cantidad de puntos que tiene el contorno, estos se enumeran uno a uno en formato (X,Y), y por el orden en que se deben ser unidos. No importa el sentido en que sean enumerados (horario o antihorario), ya que es el algoritmo de rellenado de objetos (comentado en el apartado siguiente) el que se encarga de determinar esto. Salvo en el caso del contorno de la habitación, cuyos puntos se dan en coordenadas absolutas, las coordenadas de los puntos del contorno de los demás objetos, se dan en coordenadas relativas al sistema de ejes del propio objeto. Los ejes de referencia del armario ejemplo de este capítulo, pueden verse en la *Figura 4.7.*

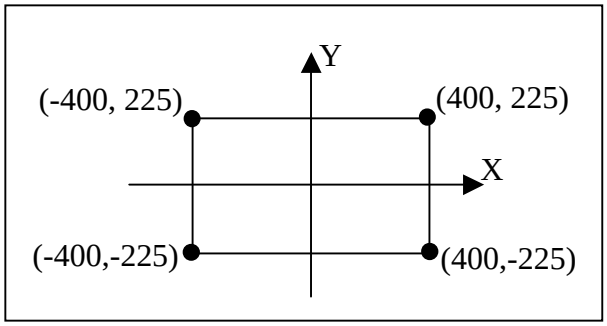


Figura 4.7: Definición de los ejes de coordenadas propios del objeto armario

No importa cuantos puntos tenga el polígono, el programa siempre lo cerrará uniendo el último punto con el primero.

4.2.3. REPRESENTACIÓN DE LA INFORMACIÓN DEL MAPA

Para la representación gráfica del mapa, tal y como se puede apreciar en la *Figura4.8.*, los objetos, al igual que las zonas externas al contorno de la habitación, se representan de color negro; mientras que las zonas por las que el robot puede realizar sus evoluciones son representadas de color blanco.

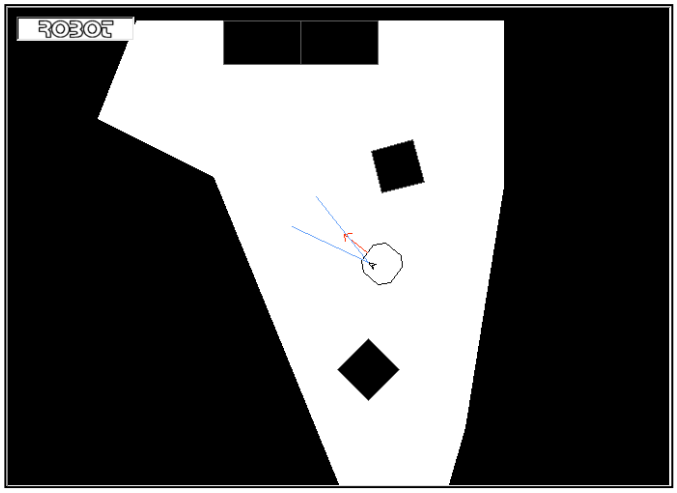


Figura 4.8: Mapa representado gráficamente en la interficie del proyecto

El proceso a seguir para rellenar un objeto de un color determinado es, al menos en principio, bastante simple, gracias a la instrucción de las MIL, MgraFill:

```
MgraFill  (GraphContId, DestImgBufId, Xstart, Ystart)
```

Donde:

- **GraphContId:** Identificador del contexto de gráficos, valor fijado a M_DEFAULT.

- **DestImgBufId:** Este parámetro especifica el identificador del "buffer" en el cual se pinta.

- **Xstart y Ystart:** Especifican las coordenadas x e y del punto semilla, relativas a la esquina superior izquierda del **DestImgBufId**. Este punto debe de estar en un área cerrada, en caso contrario, el rellenado de color se realiza hasta que los límites del buffer son encontrados. El punto semilla debe de estar dentro del "buffer" especificado, en caso contrario la operación no se realiza.

Esta instrucción realiza una rellenado similar al que ocurriría si se vierte un bote de pintura sobre una zona concreta. El punto semilla es algo así como el punto sobre el que se empieza a verter la pintura, la cual se va extendiendo hasta que encuentra un tope (el contorno de un polígono) o ya no puede extenderse más (los límites del buffer donde se pinta). Este es el motivo que obliga a representar los objetos con polígonos cerrados.

Así, el problema de pintar un objeto, reside en establecer un punto que se encuentre en el interior del polígono que lo representa, y a su vez, dentro del buffer en el que se está pintando. La solución más simple sería la de pedirle al usuario que, en el momento de definir el objeto (durante la creación del fichero objeto), determinase un punto interno que serviría de punto semilla. Desgraciadamente, tal y como se puede apreciar en la *Figura 4.9.*, esta solución es insuficiente ya que, si por desplazamientos del mapa, el objeto queda parcialmente fuera del buffer, y la zona excluida contiene el punto especificado, la parte visible del objeto no sería pintada.

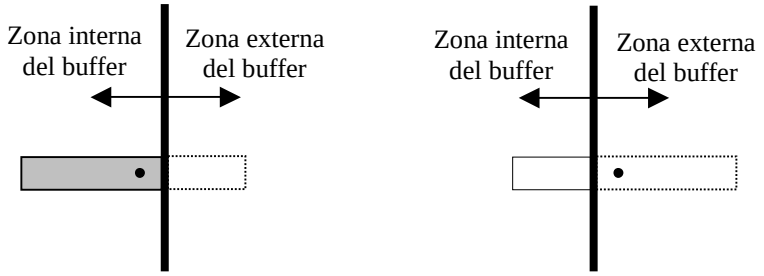


Figura 4.9 (a): **Figura 4.9 (b):**

Figura 4.9: En la Figura 4.9(a) podemos ver como una figura determinada queda parcialmente fuera del buffer, pero esta es pintada debido a que el punto semilla si queda dentro de los límites del "buffer". A causa de un pequeño desplazamiento del mapa hacia la derecha, se puede apreciar en la Figura 4.9(b) como el punto semilla queda fuera de la frontera del buffer y, por tanto, la sección de la figura que aún es visible no se pinta.

Por tanto, el sistema debe de ser la suficientemente "inteligente" como para, por un lado, buscar diferentes candidatos a punto semilla y, por el otro, verificar que este se encuentre dentro del polígono que representa el objeto, así como del buffer en el que se representa el mapa.

Para resolver el problema, en primer lugar es necesario generar los candidatos a punto semilla. El sistema utilizado es el de los "8 vecinos".

En la *Figura4.10* puede verse un ejemplo de vértice con sus 8 vecinos. Estos puntos resultan de sumar y restar una determinada cantidad a las componentes (X,Y) del vértice del polígono. La cantidad sumada no debe de ser muy grande, para que los puntos no se alejen demasiado pudiendo llegar a salirse del buffer, ni muy pequeña, para que los vecinos no caigan siempre en el contorno del polígono debido al problema del "aliasing". El problema del "aliasing" viene derivado de la discretización de una imagen en los píxeles de la pantalla.

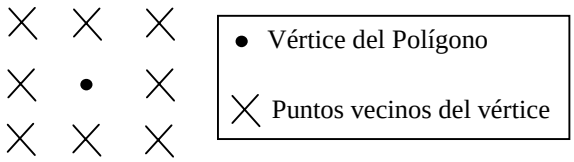


Figura 4.10: Ejemplo de vértice con sus 8 vecinos

A medida que se generan los vecinos de un vértice, se procede a verificar que se encuentren dentro del "buffer", si no es así se procede a generar el siguiente vecino, y así sucesivamente hasta que encontramos uno que pertenezca al interior del "buffer". Una vez se ha encontrado, se determina si también es interno al polígono. Si es así, el proceso finaliza, en caso contrario, se sigue buscando hasta agotar los vecinos del vértice que se esta tratando, momento en el cual se pasa al siguiente vértice. Esta situación se repite en los sucesivos vértices del polígono hasta que, se encuentra un punto semilla que cumpla las dos premisas (punto que pertenezca al buffer, e interno al polígono), o se acaban los vértices. El diagrama de flujo de la *Figura4.11* (página siguiente) muestra mucho más claramente este proceso.

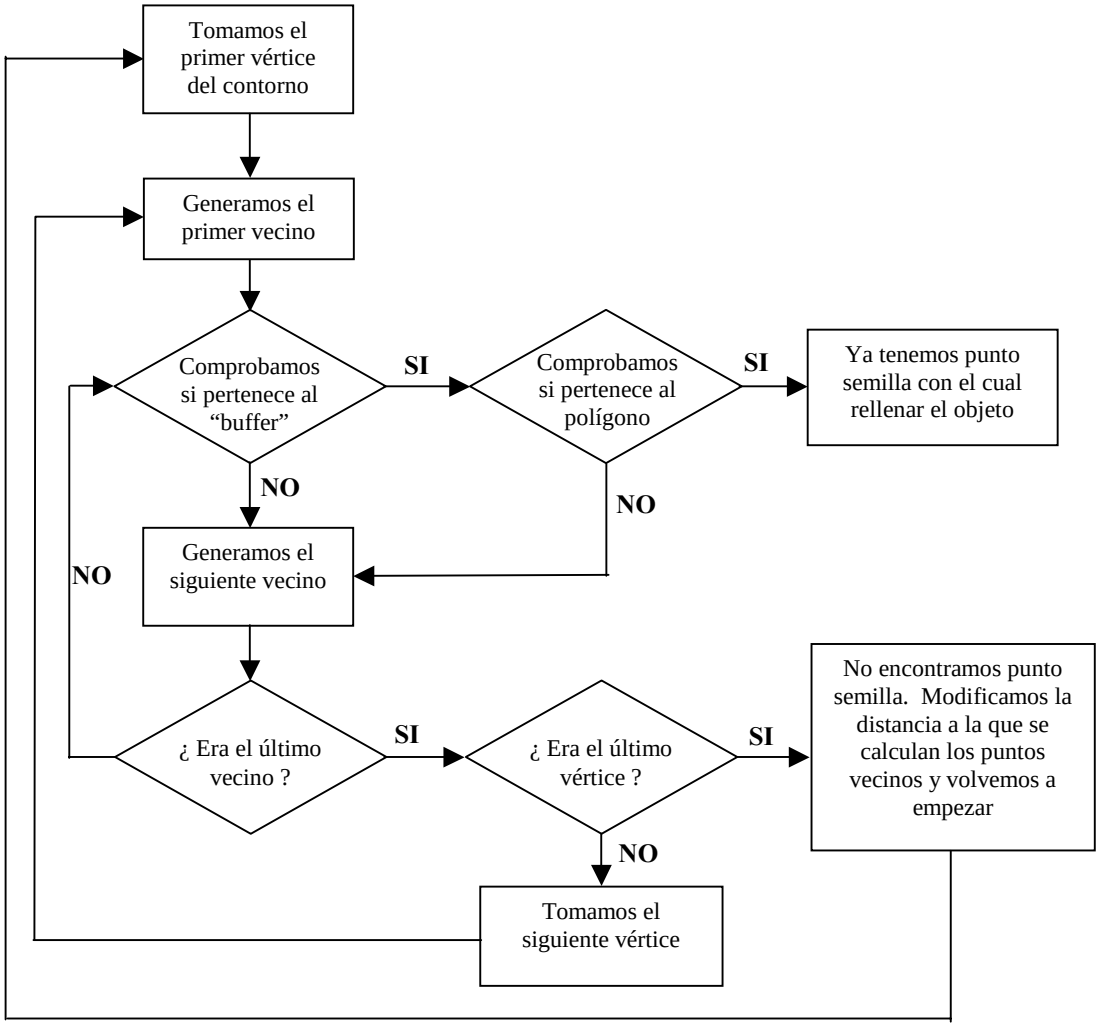


Figura 4.11: Diagrama de flujo del algoritmo de búsqueda del punto semilla

Mientras que el sistema para saber si un punto es interior o no al "buffer" es muy sencillo (se puede comprobar fácilmente transformando el punto a coordenadas de dispositivo, y verificando que sus coordenados estén entre cero y las coordenadas máximas del eje X e Y del "buffer"), comprobar si un punto se encuentra dentro de un polígono no es tan trivial, por lo que se ha optado por utilizar un algoritmo ya desarrollado, basado en la técnica de la suma de ángulos.

Para cada vértice del polígono, el ángulo diferencia entre el ángulo de este vértice y ángulo del siguiente, vistos desde el punto que esta siendo estudiado, se va añadiendo a un sumatorio. Si el resultado final de este sumatorio es ±360º, es que el punto es interior al polígono, en caso contrario el resultado del sumatorio da cero. El algoritmo utilizado, es el descrito en el artículo "An Incremental Angle Point in Polygon Test" por Kevin Weiler (WeK 94).

Mención a parte merece la representación del contorno de la habitación a causa de un problema adicional. El hecho es que, en el caso concreto del contorno, lo que debe pintarse de negro son las zonas externas al polígono que lo representa. Por tanto el problema surge a raíz que ,en la mayoría de los casos, queda más de una zona exterior entre el contorno de la habitación y los límites del "buffer" (ver *4.12.*).

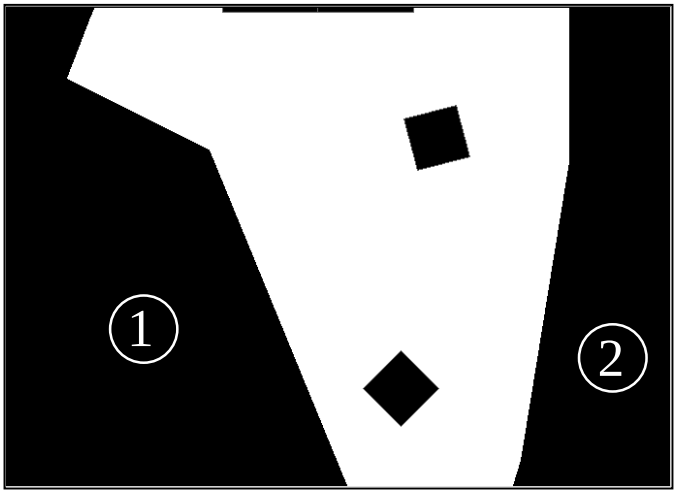


Figura 4.12: Como se puede apreciar, las zonas **1** y **2** se encuentran incomunicadas entre si, por lo que se tendrían que buscar dos puntos externos al polígono que representa el contorno de la habitación, pertenecientes cada uno de ellos a una de las regiones.

Esto significa que, debido a las limitaciones de la instrucción **MgraFill**, habría que buscar un punto externo al contorno de la habitación por cada zona que queda aislada, y puesto que pueden quedar 2 o más zonas aisladas, con el consiguiente aumento en el tiempo de cálculo para la representación, se optó por una solución más simple. Cuando se recibe la información del contorno, se pinta todo el buffer de negro, y a continuación, se pinta el contorno de la habitación de blanco para rellenarlo después del mismo color. De esta manera, las zonas exteriores al contorno de la habitación serán negras, ya que de hecho, solo el interior del polígono que representa el contorno de esta, es blanco.

4.3. SEGUIMIENTO DEL ROBOT

En este apartado se realiza un breve repaso a la geometría necesaria para la transformación de ejes de coordenadas, paso imprescindible antes de afrontar la proyección y representación del robot en el mapa, así como de las lecturas de los sonares.

4.3.1. INTRODUCCIÓN AL CÁLCULO DE TRANSFORMACIONES 2D

Son transformaciones que, aplicadas a entidades 2D, modifican la geometría pero no la topología. La forma de representar estas transformaciones es mediante matrices de tamaño n+1, donde n es el orden del espacio de representación (en nuestro caso 2D, 3x3).

Para aplicar las transformaciones a una entidad geométrica, se multiplica cada uno de los puntos que la definen, por la matriz que representa la transformación.

Básicamente se pueden realizar las siguientes transformaciones:

- Escalado
- Simetría
- Traslación
- Rotación

A continuación se da una breve explicación de cada una de ellas, debido a la gran importancia de estas transformaciones en la representación de las evoluciones del robot.

4.3.1.1. ESCALADO

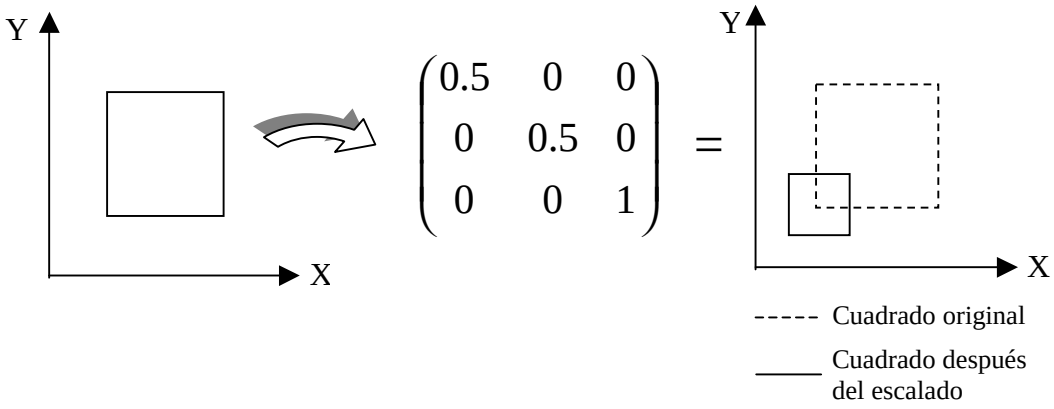
Permite modificar la escala con la que se representan las entidades en el plano. Su representación matricial es:

$$\begin{pmatrix} S_X & 0 & 0 \\ 0 & S_Y & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Sx es el valor del escalado en el eje X

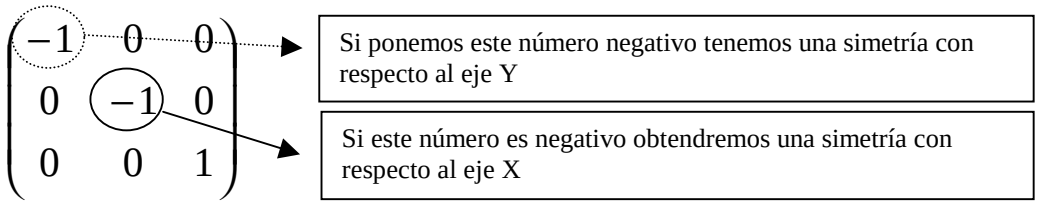
Sy es el valor del escalado en el eje Y

Si tomamos el cuadrado del ejemplo y le aplicamos la siguiente matriz de escalado, obtenemos:

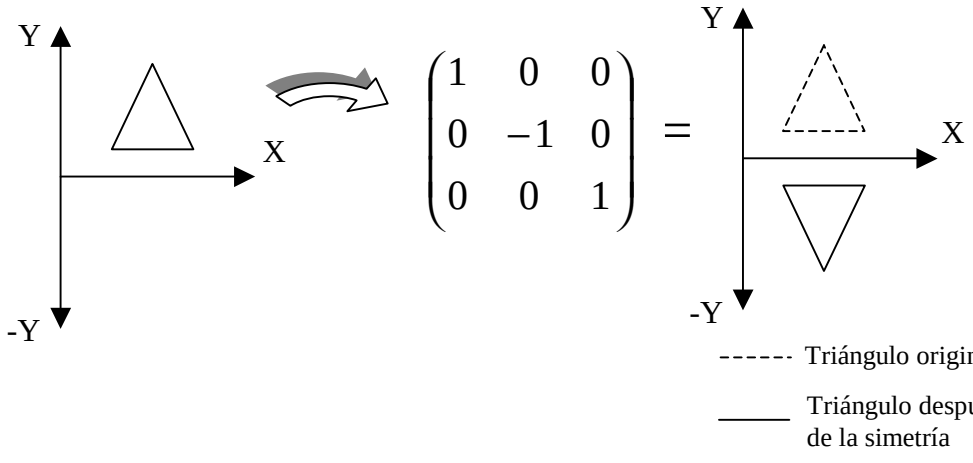


4.3.1.2. SIMETRÍA

Esta transformación permite ejecutar una simetría respecto al eje X, respecto al eje Y, o respecto a ambos ejes al mismo tiempo. Su representación matricial es:



Si se aplica una simetría con respecto al eje X al triángulo del ejemplo:



4.3.1.3 TRASLACIÓN

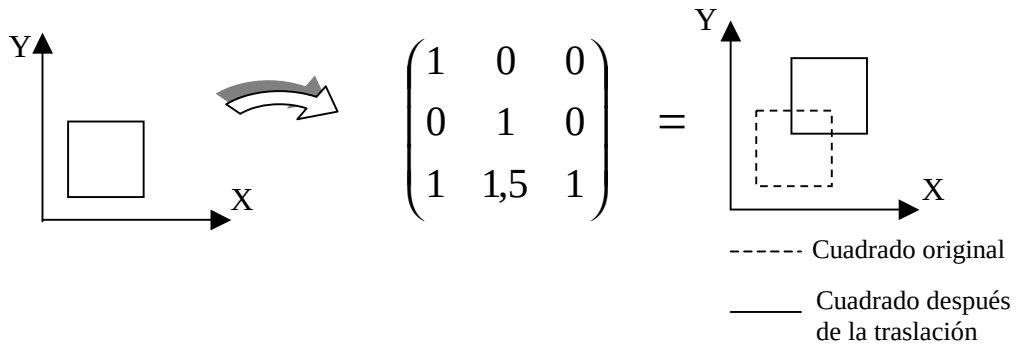
Aplicando esta transformación se consigue desplazar una entidad 2D sobre el plano de dibujo. Su matriz característica es:

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ T_X & T_Y & 1 \end{pmatrix}$$

Tx es la componente X del vector de traslación

Ty es la componente Y del vector de traslación

Si aplicamos el vector de traslación (1,1.5) al cuadrado del ejemplo obtenemos:



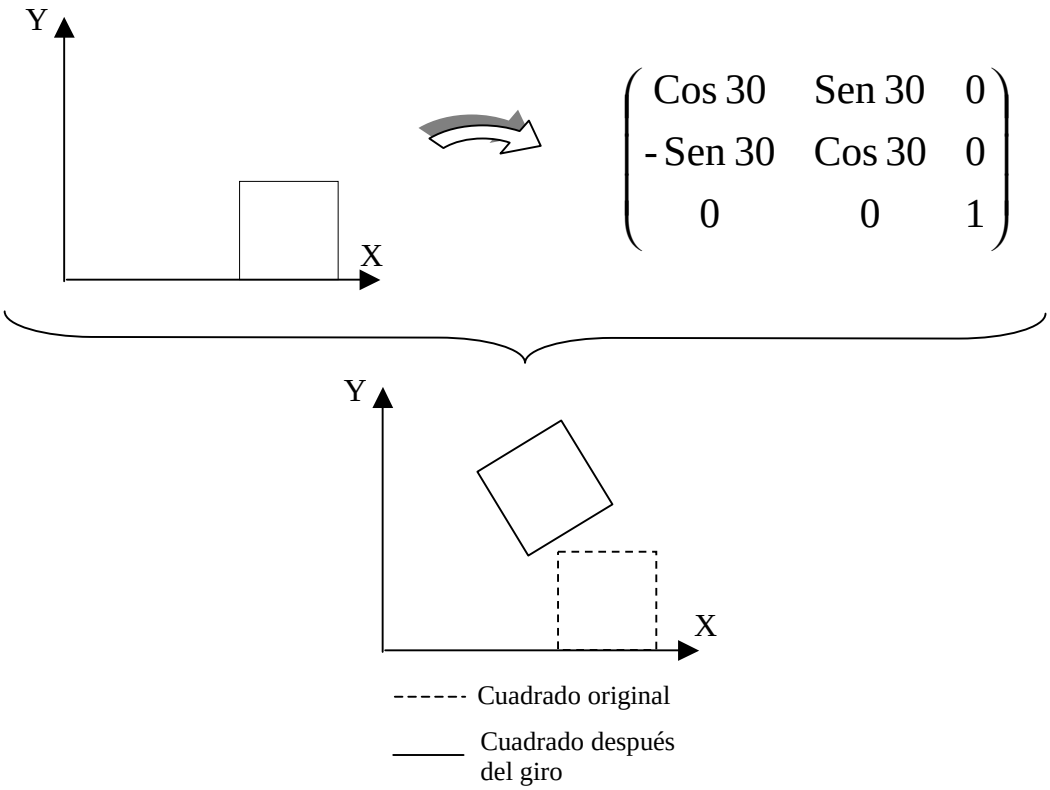
4.3.1.4. ROTACIÓN

Esta operación gira la entidad 2D α grados en torno al origen. Su expresión matricial es:

$$\begin{pmatrix} \text{Cos}\,\alpha & \text{Sen}\,\alpha & 0 \\ -\text{Sen}\,\alpha & \text{Cos}\,\alpha & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

α es el ángulo en que queda la figura después de la rotación

Supongamos que giramos 30 grados el cuadrado del ejemplo:


Y
X
$\begin{pmatrix} \text{Cos } 30 & \text{Sen } 30 & 0 \\ -\text{Sen } 30 & \text{Cos } 30 & 0 \\ 0 & 0 & 1 \end{pmatrix}$
Y
X
Cuadrado original
Cuadrado después del giro

4.3.2. COMPOSICIÓN DE TRANSFORMACIONES GEOMÉTRICAS 2D

Para componer transformaciones geométricas complejas se multiplican las matrices que representan las transformaciones individuales, en el orden en que deben sucederse.

$$M_{Resultado} = M_1 \cdot M_2 \cdot \ldots \cdot M_n$$

Un detalle a tener muy en cuenta es que el producto de matrices NO es conmutativo, por lo cual, no es lo mismo aplicar una rotación y luego una traslación (*Figura 4.3*), que primero trasladar y luego rotar (*Figura 4.4*).

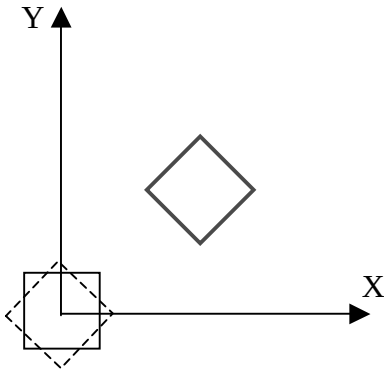


Figura 4.13: Ejemplo de giro y traslación

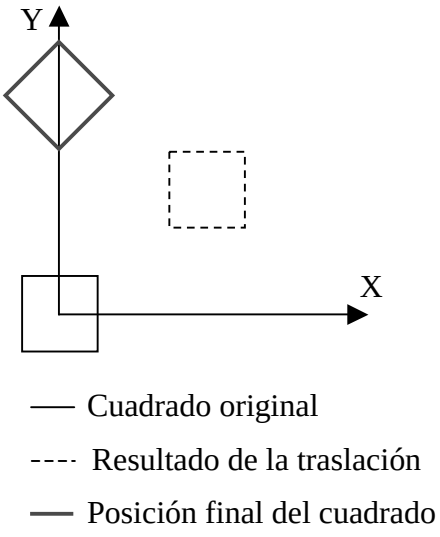


Figura 4.14: Ejemplo de traslación y giro

4.3.3. CÁLCULO DE LA MATRIZ DE CAMBIO DE BASE

Aunque las evoluciones del robot han de ser representadas en pantalla, debe tenerse en cuenta que el sistema de referencia de la misma no es el más adecuado para los objetivos del proyecto.

Como se puede apreciar en la *Figura4.15* (página siguiente), el sistema de referencia de pantalla se encuentra en la esquina superior izquierda, con el eje de ordenadas orientado hacia abajo y el de abcisas hacia la derecha. Las unidades que utiliza son los píxeles (unidad mínima que puede representar un monitor).

El sistema de referencia que interesa para el proyecto, se encuentra en la esquina inferior izquierda, con el eje de ordenadas orientado hacia arriba y el de abcisas hacia la derecha. Las unidades con las que trabaja Saphira son los milímetros y de hecho, son las unidades con las que se trabajará.

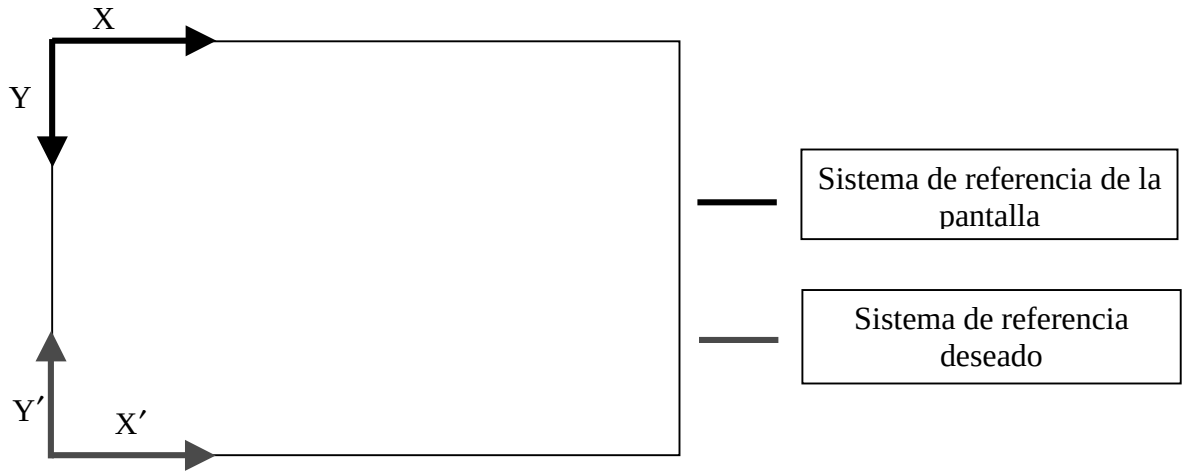
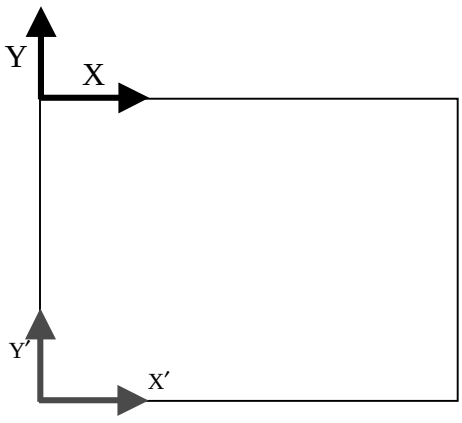


Figura 4.15

Lo más conveniente, sería disponer de una función que, dado un punto de mundo cualquiera (expresado en milímetros y en el eje de coordenadas rojo), realizase las transformaciones 2D necesarias para representar este punto en coordenadas de pantalla. De esta manera no tenemos que preocuparnos de definir los objetos en la orientación del sistema de referencia de la pantalla, ni de trabajar con píxeles, simplemente dando los puntos en el sistema de coordenadas que nos interesa, y en milímetros, sabemos que se encargará de traducir sus coordenadas para que se pinten correctamente en pantalla.

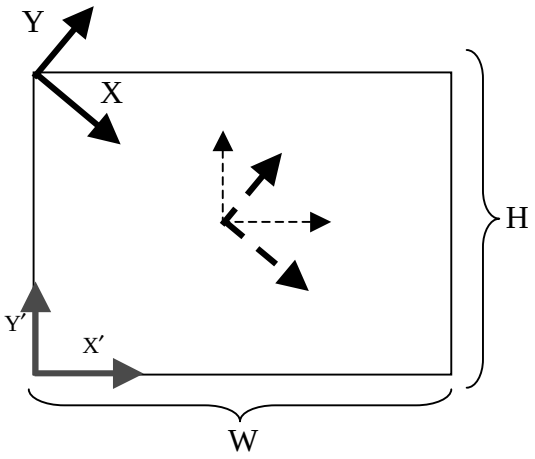
La función en cuestión debe calcular una matriz de cambio de base que reúna todas las operaciones necesarias para la transformación de un sistema de referencia al otro. Dichas operaciones, representadas cada una de ellas por matrices, se calculan de la siguiente manera:



Simetría

Se Realiza una simetría con respecto al eje X, de manera que el eje de ordenadas toma la orientación deseada

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$



Giro con respecto al centro del mapa

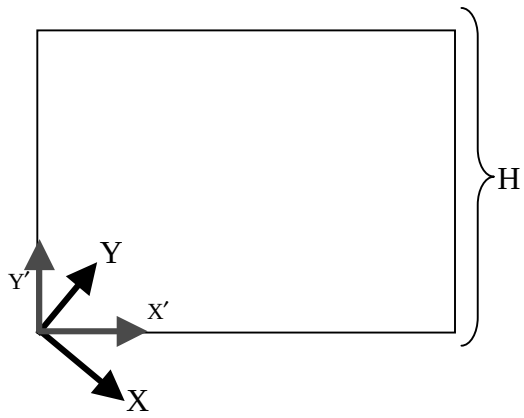
Al usuario le puede interesar girar la vista del mapa, pero no respecto al origen, sino con respecto al centro del mapa. Esto quiere decir que debe trasladarse la base al centro del mapa, girar los grados necesarios, y devolver la base a donde estaba:

Sistema de referencia trasladado al centro del mapa

Sistema de referencia girado α grados

Sistema de referencia devuelto al origen

$$M_{\substack{\text{Giro}\\ \text{repecto al}\\ \text{centro}}} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ W/2 & -H/2 & 1 \end{pmatrix} \cdot \begin{pmatrix} \text{Cos}\,\alpha & \text{Sen}\,\alpha & 0 \\ -\text{Sen}\,\alpha & \text{Cos}\,\alpha & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -W/2 & H/2 & 1 \end{pmatrix}$$



Traslación a la esquina inferior izquierda

Una vez girada al ángulo indicado, se traslada la base a la esquina inferior izquierda.

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -H & 1 \end{pmatrix}$$

Por tanto, siguiendo las reglas para la composición de trasformaciones geométricas (capítulo 4.3.2.), la matriz de cambio de base se expresa:

Giro con respecto al centro del mapa

$$M_{Resultante} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \overbrace{\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ W/2 & -H/2 & 1 \end{pmatrix} \cdot \begin{pmatrix} \text{Cos}\,\alpha & \text{Sen}\,\alpha & 0 \\ -\text{Sen}\,\alpha & \text{Cos}\,\alpha & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -W/2 & H/2 & 1 \end{pmatrix}}^{} \cdot$$

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -H & 1 \end{pmatrix} \cdot \begin{pmatrix} \text{Escalado} & 0 & 0 \\ 0 & \text{Escalado} & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

4.3.4. POSICIONAMIENTO DEL ROBOT

El gráfico que representa el robot no deja de ser una entidad 2D, la cual puede ser representada donde sea necesario mediante las oportunas transformaciones geométricas. Una vez obtenida la matriz de cambio de base, y con los conocimientos necesarios para la rotación, traslación, y demás operaciones sobre entidades geométricas 2D, solo falta conocer la posición y orientación del robot real en cada momento, a fin de poder realizar las transformaciones necesarias.

Para obtener dicha información, Saphira pone a disposición del usuario la estructura *sfRobot*. Esta estructura contiene los siguientes campos:

Campos de sfRobot	Unidades	Descripción
x, y, th	mm, mm, grados	Localización del robot en coordenadas del robot; siempre (0,0,0)
ax, ay, ath	mm, mm, grados	Localización global del robot
tv, mtv	mm/seg	Velocidad actual y máxima respectivamente
rv, mrv	grados/seg	Velocidad rotacional actual y máxima respectivamente
leftv, rightv	mm/seg	Velocidades de las ruedas izquierda y derecha
Status	int STATUS_STOPPED STATUS_MOVING STATUS_NOT_CONNECTED STATUS_NO_HIGH_POWER	Estatus del robot: Robot parado Robot Moviendose Cliente no conectado Motores del robot calados

Battery	1/10 volt	Tensión de la batería
Bumpers	Int	Estado de la batería
Ptu	usens	Cabeceo de la unidad Pan/Tilt
Diginput	int	Estado de la entrada digital
Digoutput	int	Estado de la salida digital
motor_packet_count sonar_packet_count vision_packet_count	cómputos por segundo	Información del paquete de comunicaciones

Antes de utilizar los datos de la estructura *sfRobot*, se deben tener en cuenta 2 factores. En primer lugar, el sistema de referencia que utiliza Saphira (ver *Figura 4.16*), no es el mismo que el definido en este proyecto. Es posible utilizar transformaciones geométricas 2D (un simple giro de 90º en el sentido horario bastaría) para igualar los sistemas de referencia, pero es mucho menos costoso computacionalmente hacer la asignación de los valores de la siguiente forma:

```
PosX = -sfRobot.ay;
PosY = sfRobot.ax;
Heading = sfRobot.ath;
```

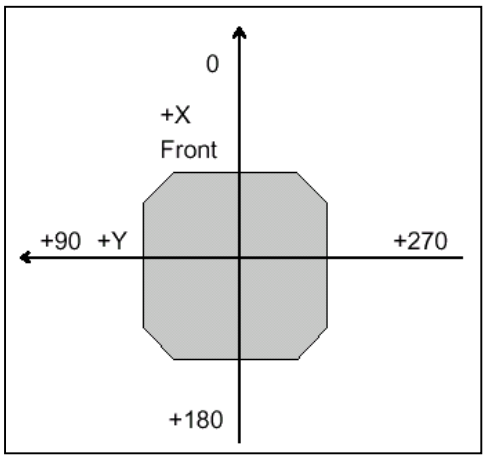


Figura 4.16: orientación de la base en Saphira

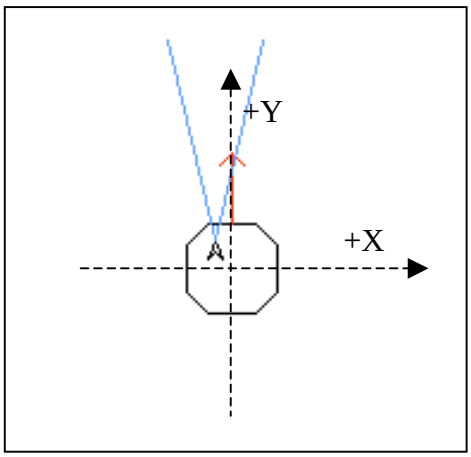


Figura 4.17: Orientación de la base en este proyecto

En segundo lugar, en el momento de la conexión, y a no ser que se le indique lo contrario, Saphira presupone que la posición del robot es X=0, Y=0 y Th=0. Esto, especialmente en el caso de los mapas predefinidos, puede producir algún conflicto, ya que en la mayoría de los casos se dará por supuesto que el robot esta en una posición determinada que no tiene porque ser el origen de coordenadas. En este estado de desarrollo del proyecto puede no ser un problema, pues sumando (o restando según convenga) las coordenadas de inicio del robot a las que va suministrando Saphira en la estructura *sfRobot*, el problema queda resuelto; pero en un futuro, cuando el robot se autoposicione, será necesario decirle a Saphira cual es la posición real que el robot ocupa

en ese momento, puesto que por pequeños errores acumulados de odometría, la posición reflejada por el robot no será la correcta. La instrucción que permite modificar esto es *sfMoveRobot (x,y,th)*, donde x,y,th son la posición y orientación absoluta que Saphira debe presuponer para el robot a partir de ese instante.

Con la información que se obtiene de *sfRobot*, se dispone de todos los datos necesarios para posicionar el robot sobre el mapa. El robot virtual, al igual que los demás objetos del mapa, tiene sus puntos definidos en coordenadas propias (ver *Figura 4.17.*), por lo que se debe posicionar el sistema de ejes del robot en el lugar adecuado para representar las evoluciones del robot real. Las operaciones a realizar son:

-Rotar el robot virtual para que este orientado en la misma dirección que el robot real

$$\begin{pmatrix} \text{Cos (Heading)} & \text{Sen (Heading)} & 0 \\ -\text{Sen (Heading)} & \text{Cos (Heading)} & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

-Una vez rotado, desplazar el robot virtual a la posición absoluta que ocupa.

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ \text{PosX} & \text{PosY} & 1 \end{pmatrix}$$

A continuación, y para finalizar, es necesario pasar todos los puntos que componen el robot por la matriz de Cambio de base (apartado 3.3.2.), obteniendo el valor de los puntos que deben de pintarse en pantalla.

El proceso de representación del robot es el último antes de pasar a obtener las lecturas de los sonares. Durante el proceso de representación de la escena sucede que, debido a la velocidad con la que se ejecuta, puede apreciarse como los objetos se van generando uno a uno hasta que se han generado todos. Aún en el mejor de los casos, llega a percibirse un continuo y molesto parpadeo del robot mientras este se desplaza, siendo este hecho mucho más evidente cuando se trata de repintar toda la escena.

Para subsanar este inconveniente, la técnica más utilizada es la del “off-screen buffer”, que consiste en utilizar un buffer adicional, no visible, en el que se compone toda la escena antes de ser mostrada.

4.4. REPRESENTACIÓN DE LA INFORMACIÓN DE LOS SONARES

4.4.1. COMO ALMACENA EL ROBOT LA INFORMACIÓN DE LOS SONARES

Las lecturas que los sonares del robot reciben de su entorno son almacenadas en tres "buffers" internos. En un "buffer" se almacenan las lecturas de los sonares delanteros y traseros, mientras que existe un "buffer" independiente para las lecturas de los sonares laterales izquierdos, y otro para los laterales derechos.

El porqué de almacenar las lecturas en "buffers" diferentes, responde a las diferentes necesidades que puede tener el software de control del robot. Los sonares frontales y traseros, orientados en la dirección de la traslación, sirven para la detección de obstáculos, pero la definición de los sonares no es muy buena, por lo que es prácticamente imposible definir la forma del mismo. Los sonares laterales, de alguna manera, también son útiles en la detección de obstáculos (sobretodo en los giros a un lado y otro); pero su principal objetivo es perfilar características para los algoritmos de reconocimiento. Esto se debe a que en la mayoría de los casos el robot se desplaza paralelo a paredes, por lo que mientras se van acumulando las lecturas, se pueden ir extrayendo características.

Cada uno de estos tres "buffers" es una estructura *cbuf*:

```
#define CBUF_LEN 200

typedef struct
{
   int start;              /*Puntero a buffer interno */
   int end;                /* Puntero a buffer interno*/
   int limit;              /*Tamaño actual del buffer*/
   float xbuf [CBUF_LEN]
   float ybuf [CBUF_LEN]
   int valid [CBUF_LEN]
} cbuf;

cbuf *sraw_buf,  *sr_buf, *sl_buf
```

Figura 4.18: Estructura de datos, en C, que almacena la información de los sonares

Los vectores xbuf e ybuf almacenan las coordenadas x e y respectivamente, de las lecturas de los sonares, mientras que el vector de enteros *valid* determina si la lectura

correspondiente es válida o no. El porqué de la necesidad de un vector que nos indique si la lectura almacenada es válida o no, precisa de una explicación más profunda.

Aunque los tres "buffers" son circulares, es decir, cuando se alcanza el límite de almacenamiento las nuevas lecturas sustituyen a las más viejas (listas First Input First Ouput o FIFO), debido a las diversas aplicaciones que puede tener los diferentes "buffers", la manera en que estos almacenan las lecturas también difiere de unos a otros.

El "buffer" frontal (sraw_buf) acumula una lectura cada vez que uno de los sonares es disparado, obtenga o no respuesta. Si se obtiene respuesta, el flag *valid* toma el valor 1, en caso contrario toma el valor 0. Esto provoca que, si el robot esta tomando 20 lecturas por segundo y el "buffer" tiene una profundidad de 30 elementos (CBUF_LEN=30) , este se encontrará completamente vacío en tan solo 1,5 segundos si no hay nada alrededor del robot.

Muy al contrario, los dos "buffers" laterales (sr_buf y sl_buf) solo acumulan lecturas si estas son positivas, por lo que en este caso, el flag *valid* será siempre 1. De este modo, las lecturas de los sonares laterales permanecen más tiempo en los "buffers", permitiendo una mejor extracción de características a partir de estas.

Es importante saber que la profundidad de los "buffers" (valor de CBUF_LEN), es modificable mediante las instrucciones de C:

```
void sdSetFrontBuffer (int n)
void sfSetSideBuffer (int n)
```

Estas funciones fijan la profundidad de los "buffers" frontales/traseros y los laterales respectivamente. Si el valor dado al argumento es mayor que cero, se asigna ese valor a la profundidad de los "buffers", si, en caso contrario, el argumento tiene valor 0, las funciones ponen el vector de flags *valid* a cero.

4.4.2. REPRESENTACIÓN DE LA INFORMACIÓN DE LOS SONARES

Las lecturas de los sonares almacenadas en los "buffers" de Saphira, están en coordenadas de la base solidaria al robot, por lo que se debe realizar un cambio de base al sistema de referencia del mundo para poder representarlas en el mapa del Navegador. La orientación de la base es la estándar de Saphira (ver *Figura 4.19*, en la página siguiente).

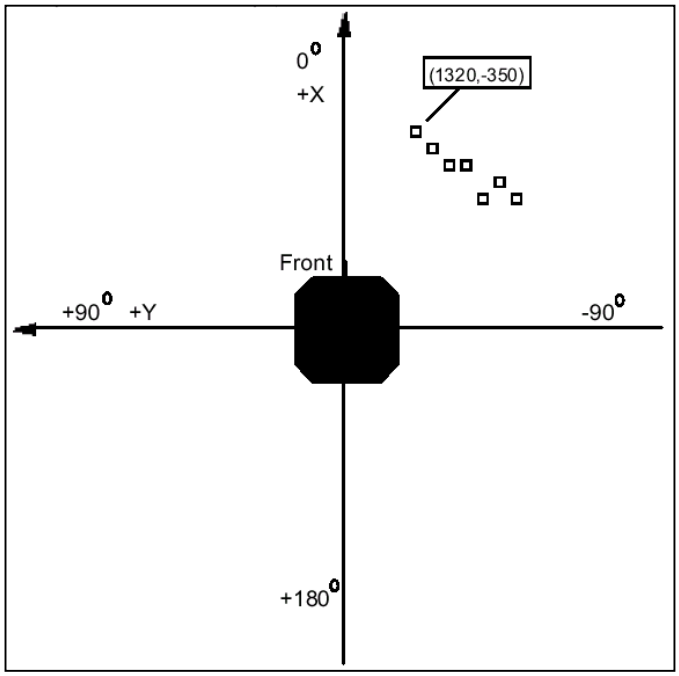


Figura 4.19: Ejemplo de lecturas de sonares en Saphira

Las operaciones a realizar son las mismas que en el caso del control del robot (capítulo 4.3.4.), es decir, si se toma la lectura de la imagen como ejemplo:

```
CoordX=1320
CoordY=-350
```

Se realiza el cambio oportuno en las asignaciones de las coordenadas:

```
NewCoordX=-CoordY
NewCoordY=CoordX
```

Obteniendose:

```
NewCoordX=350
NewCoordY=1320
```

Ahora las coordenadas de la lectura están en función de la base solidaria al robot , pero con la orientación que nosotros tenemos definida en el proyecto. El siguiente paso será transformar las coordenadas de la lectura, del sistema de referencia del robot al sistema de referencia de mundo. De nuevo, los pasos a realizar son los mismos que en el caso del robot:

$$\begin{pmatrix} \text{Cos (Heading)} & \text{Sen (Heading)} & 0 \\ \text{- Sen (Heading)} & \text{Cos (Heading)} & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ \text{PosX} & \text{PosY} & 1 \end{pmatrix}$$

Donde Heading es el ángulo que forma la base solidaria al robot con la referencia del mundo; y (PosX, PosY) son las coordenadas del robot en la referencia de mundo.

Las lecturas ya están expresadas en coordenadas del sistema de referencia de mundo. Transformando de nuevo sus componentes, esta vez con la matriz de cambio de base, las coordenadas serán las necesarias para pintar las lecturas en la posición que deben ocupar en el mapa.

Aunque internamente se almacenen todas las lecturas (o no) de los sonares para su posterior análisis, la realidad es que mostrar todas estas lecturas puede ser relativamente

costoso computacionalmente hablando. Como ya se comentó en el capítulo anterior, si tenemos que cada sonar realiza 20 lecturas por segundo, un sencillo cálculo revela que, 16 sonares por 20 lecturas cada segundo, arroja una cifra de 320 lecturas por segundo. Teniendo en cuenta que deben ser tratadas antes de ser representadas en pantalla, se deduce que estas tardarán un cierto tiempo en ser representadas en pantalla, ralentizando la actualización del mapa. De hecho muchas de estas lecturas estarán en posiciones muy similares, por lo que a efectos estéticos el usuario solo vería una nube muy densa de puntos. Para prevenir esto, hay un parámetro llamado LISTA_MAX que permite determinar la cantidad máxima de lecturas de sonares que se representan en pantalla. Esta lista, es una lista FIFO (First Input First Output), o lo que es lo mismo las primeras lecturas en aparecer son también las primeras en desaparecer.

En la *Figura 4.20.* se puede apreciar como queda representada toda la información una vez que se ha finalizado el proceso de construcción del mundo, posicionamiento del robot y representación de las lecturas de los sonares.

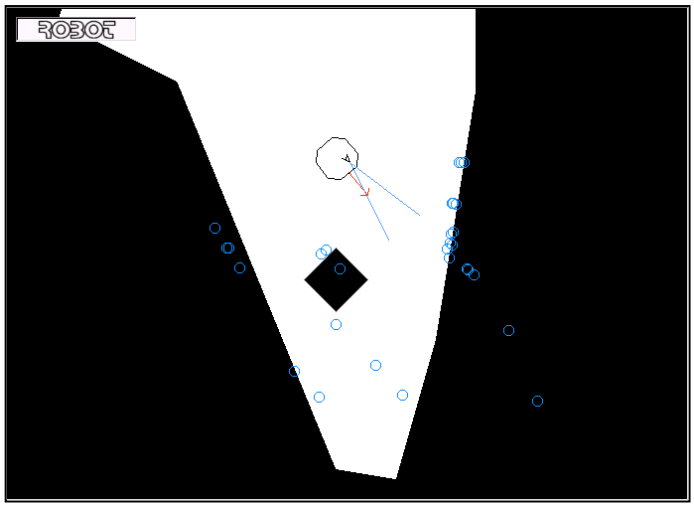


Figura 4.20: Representación de los sonares en la interfaz del proyecto para un mapa pregenerado

4.5. CONEXIÓN DE LA CÁMARA DE NAVEGACIÓN

Para facilitar la tarea al usuario durante el guiado manual del robot, se utiliza una de las cámaras dedicadas a estéreo-visión para que facilite información visual del entorno.

Para poder captar la información enviada por las cámaras, el PC de navegación esta provisto de una tarjeta de adquisición de video Matrox Meteor. Como ya se comento en el capítulo de descripción de Hardware y Software (apartado 3.2.1. Librerías MIL), para comenzar a adquirir imágenes, es necesario inicializar los recursos del hardware y abrir los canales de comunicación con la tarjeta. Ya que se ha de “alocatar” el sistema Meteor, ¿por qué no reservar memoria para todos los “buffers” necesarios en el mismo sistema (Meteor), en lugar de reservarlos en otro sistema (VGA)?.

El principal motivo es la portabilidad del programa a otros sistemas (PC's). En caso de querer ejecutar el programa en otro ordenador que no dispusiera de una tarjeta gráfica Meteor, sería imposible arrancar el programa para realizar un seguimiento del robot o controlar sus comportamientos, puesto que no podrían ser alocatados los "buffers" mínimos para su funcionamiento.

Por esta razón se decidió mantener 2 sistemas independientes. Por un lado el sistema encargado de plasmar la información en pantalla (tarjeta gráfica VGA), incluyendo el seguimiento del robot, representación de la información de los sonares, y demás información. Por otro lado el sistema encargado de la adquisición de imágenes por medio de las cámaras del robot

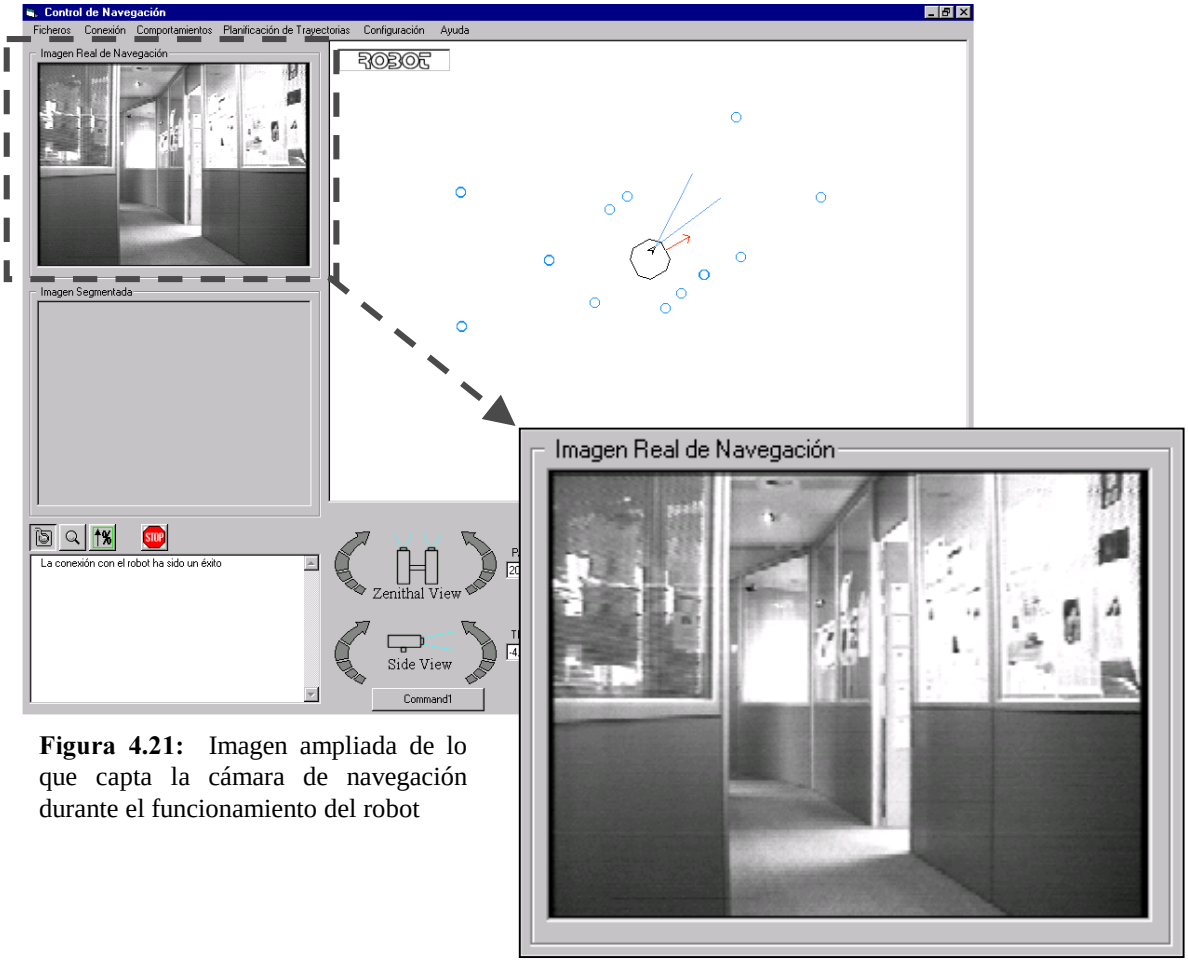


Figura 4.21: Imagen ampliada de lo que capta la cámara de navegación durante el funcionamiento del robot

Además de abrir los canales de comunicación con la tarjeta en sí, también es necesario reservar memoria en el sistema para un digitalizador, de manera que puedan ser usadas las funciones MIL de digitalización. La función que realiza esta operación es:

```
MdigAlloc(SystemId, DigNum, DataFormat, InitFlag, DigIdPtr)
```


Donde:

`SystemId`: Es el identificador del sistema en el que será reservada la memoria para el digitalizador.

`DigNum`: Especifica el número del digitalizador para el que se esta reservando memoria. Fijado a M_DEFAULT.

`DataFormat`: Especifica el tipo de formato de datos, o el nombre del fichero en el cual se puede encontrar el formato de datos, del dispositivo de entrada. En este caso concreto se ha definido el formato de datos "M_CCIR" (norma de video estándar europeo monocromático PAL), que recoge los datos del dispositivo de entrada (cámara) con el siguiente formato: 768x576 píxeles, 8 bits, 14.8MHz, análogico

`InitFlag`: Establece el tipo de inicialización que se desea realizar en el digitalizador. Este parámetro debe de ser fijado M_DEFAULT.

`DigIdPtr`: Es el puntero al identificador del digitalizador.

Después de reescalar las imágenes de manera que ocupen todo el espacio disponible de representación (ver *Figura4.21*), se reserva espacio en memoria para el buffer especificando en sus atributos M_GRAB. Este atributo habilita la grabación de datos en un “buffer”.

El código en C++ que se encarga de realizar todos los pasos necesarios para la adquisición de imágenes es el que se muestra en la *Figura4.22* (página siguiente).

La adquisición de imágenes se realiza con una frecuencia determinada, establecida por el usuario en el menú de opciones correspondiente. No mostrar una secuencia de video en tiempo real, se justifica por el hecho que la carga de trabajo que provoca en el sistema, impide un funcionamiento normal del mismo.

```
DLL_EXPORT void STD ConectarCamaraMIL (HWND WindowHandle,MIL_ID SysMETEOR,
                                       MIL_ID *disp, MIL_ID *camara, MIL_ID
                                       *digitalizer,long alto,  long ancho)
{
// Alocatamos el Display para la imagen
    MdispAlloc (SysMETEOR,M_DEFAULT,"M_DEFAULT",M_DEFAULT,&*disp);
// Alocatamos el digitalizador
    MdigAlloc (SysMETEOR,M_DEFAULT,"M_CCIR",M_DEFAULT,&*digitalizer);
// Selecciona el canal de entrada activo para el digitalizador
    MdigChannel(*digitalizer, M_CH0);
// Escalamos la imagen de la camara en el eje X y en el eje Y
    MdigControl(*digitalizer, M_GRAB_SCALE_X, M_FILL_DISPLAY);
    MdigControl(*digitalizer, M_GRAB_SCALE_Y, M_FILL_DISPLAY);
// Preguntamos al digitalizador el tamaño de la imagen que adquiere
    MdigInquire(*digitalizer, M_SIZE_X, &ancho);
    MdigInquire(*digitalizer, M_SIZE_Y, &alto);
// Alocatamos el buffer
    MbufAlloc2d(SysMETEOR,alto,ancho,8+UNSIGNED,
                    M_IMAGE+M_GRAB+M_DISP,&*camara);
// Seleccionamos la ventana donde se mostrarán las imágenes
    MdispSelectWindow (*disp,*camara,WindowHandle);
}
```

Figura 4.22: Código en C++ que se encarga de comenzar la adquisición y muestreo de imagenes

5 CONTROL DE LOS SISTEMAS MECÁNICOS DEL ROBOT

5.1. INTRODUCCIÓN

En este capítulo se tratan los distintos modos de control de los sistemas mecánicos del robot así como los problemas derivados de su conexión en red, etc... Concretamente los sistemas que se tratan son el control del Pan&Tilt y el del robot en si mismo.

5.2. PAN&TILT

Para la orientación del sistema de estéreo-visión se dispone de un Pan&Tilt, el control del cual se realiza gracias a las librerías escritas en C suministradas por el propio fabricante. Estas librerías permiten implementar desde simples programas de control, hasta programas mucho más complejos destinados al seguimiento de objetos, búsqueda de landmarks, etc...

La conexión básica del Pan&Tilt se realiza, tal y como se muestra en la *Figura 6.1*, mediante el puerto serie de un PC (RS-232) por el que se envian las instrucciones al controlador del Pan&Tilt, auténtico encargado de interpretarlas de manera que el Pan&Tilt ejecute las acciones apropiadas. El problema surge cuando el ordenador que debe controlar el Pan&Tilt no esta conectado a él, sino a otro ordenador de una red, haciendo necesario el desarrollo de un programa de enlace entre los dos dispositivos.

CONTROL DEL PAN&TILT VÍA TCP/IP

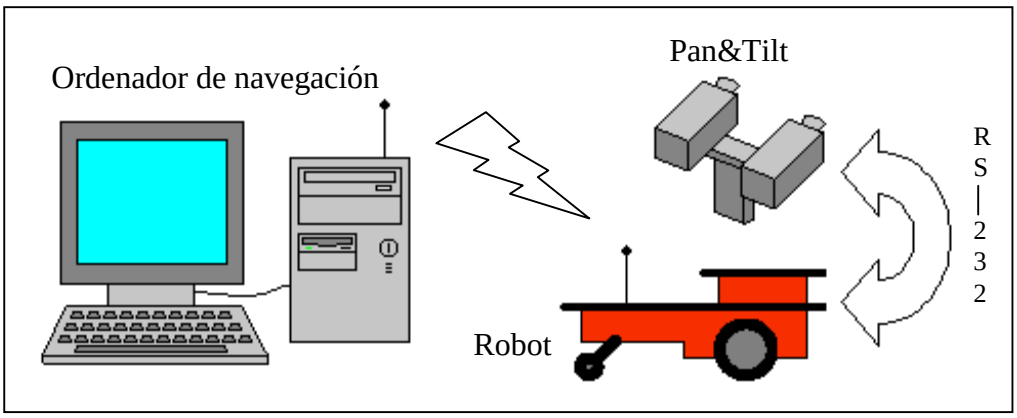


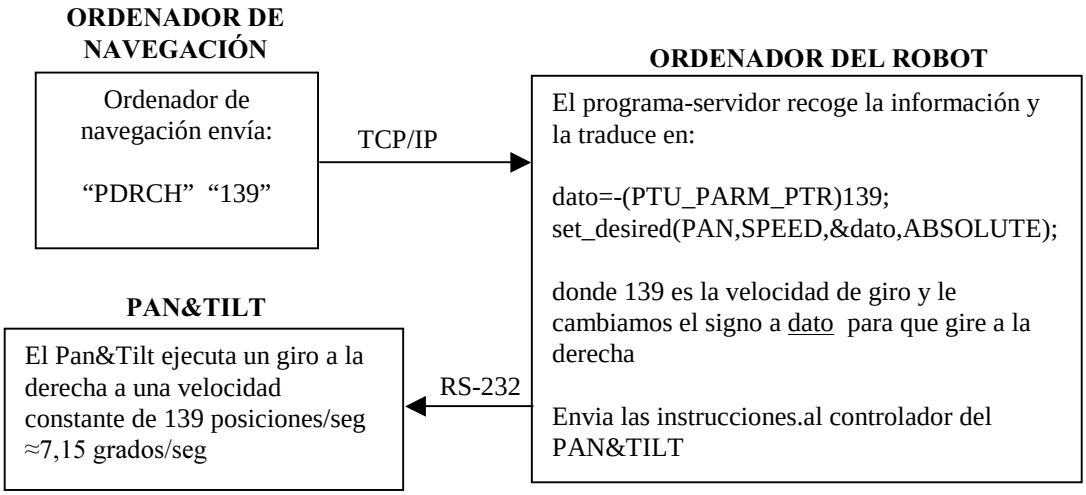
Figura 5.1:. Configuración de las conexiones del robot y Pan&Tilt

La *Figura 5.1* muestra cual es la configuración hardware disponible. Como se puede observar, el Pan&Tilt esta conectado a un PC (el interno del robot) el cual, a su vez, está conectado a una red de ordenadores vía Ethernet. Es un ordenador cualquiera de esta red (ordenador de Navegación) el que se encarga de controlar el robot, y por consiguiente, del control del Pan&Tilt.

Las librerías del fabricante no contemplan la posibilidad de controlar el Pan&Tilt vía TCP/IP, por lo que se optó por realizar una conexión entre los dos ordenadores tipo cliente/servidor, similar a la que utiliza Saphira para el control del robot. Pero la implementación de los Sockets permitía varias soluciones:

5.2.1. PRIMERA SOLUCIÓN

En un primer momento se pensó que poner el programa-servidor en el ordenador de a bordo del robot sería lo más adecuado. Su misión sería la de estar continuamente a la espera de una conexión, y una vez que esta se realizase, interpretaría unos códigos definidos a priori (durante la implementación del código) enviados por el programa-cliente (ejecutándose en el ordenador de navegación). Estos códigos serian traducidos por el mismo servidor en instrucciones propias de las librerías del Pan&Tilt, que al ejecutarse en el mismo ordenador en el que este está conectado, podrían ser llevadas a cabo sin dificultad. Esquemáticamente sería:



Esta manera de afrontar el problema presentaba un rendimiento muy bueno en cuanto a tiempo de respuesta, ya que la cantidad de datos enviados vía Ethernet era la mínima posible (tan solo 2 envíos por cada orden). Desgraciadamente cuenta con 2 grandes inconvenientes:

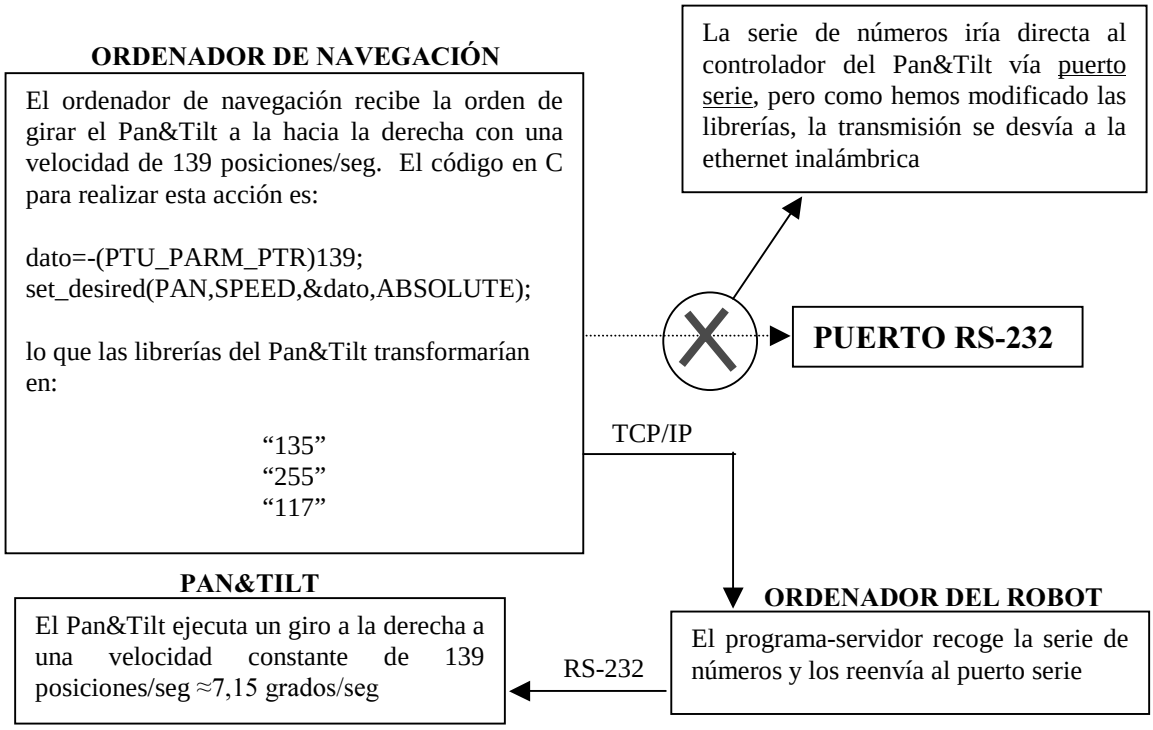
- Por un lado, todos los programas escritos hasta el momento para el control del Pan&Tilt, así como futuras implementaciones, deberían reestructurarse profundamente a fin de utilizar unas funciones que se encargasen de generar los códigos predefinidos y enviarlos, en lugar de las funciones propias de las librerías suministradas por el fabricante.

- El programa servidor debería codificar todas y cada una de las funciones disponibles del Pan&Tilt que tuviera que ser capaz de interpretar. Codificar un juego sencillo de instrucciones para girar el Pan y el Tilt mediante control por velocidad, no es nada complejo. Sin embargo, la tarea se complica enormemente cuando de lo que se trata es de codificar todas posibilidades de control por aceleración, por posición, por velocidad, consultas de la posición del Pan, consultas de la posición del Tilt, etc... que presentan las librerías de control del fabricante.

5.2.2 SEGUNDA SOLUCIÓN

Después de un estudio del código de las librerías suministradas por el fabricante, se observó que las funciones de bajo nivel de estas se encargaban de transformar las ordenes de control en series de números que eran enviados secuencialmente por el puerto serie (RS-232), interpretándolas luego el controlador del Pan&Tilt.

Teniendo en cuenta esto, se planteó la posibilidad de modificar las librerías originales de manera que, una vez obtenidas las series de números, en lugar de ser enviados al puerto serie, estos serian redireccionados, vía ethernet inalámbrico, al programa-servidor. En este caso, el programa-servidor, que tal como sucedía en la primera solución esta funcionando en el ordenador de a bordo del robot, no tiene que interpretar ningún código, simplemente envía los números recibidos directamente al puerto serie. En el siguiente ejemplo se puede ver con más claridad:



Mientras que en la primera solución, para enviar una orden, solo se transmitían 2 cadenas de texto (una para especificar la orden y otra para la velocidad), como se puede observar en el ejemplo, en esta segunda solución las cadenas a enviar vía Ethernet para la misma orden son 3, más 6 cadenas de texto correspondientes a la comunicación de la posición del Pan&Tilt (3 cadenas para consultarla, y 3 más para comunicarla), que en el primer caso no se consultaba. Esto hace un total de 9 cadenas de texto enviadas vía Ethernet, con el consiguiente aumento de tiempo para enviar una simple orden. En contraposición las ventajas son:

- Todos los programas diseñados para funcionar con el Pan&Tilt conectado directamente al puerto serie del ordenador, funcionarían en red añadiendo una simple línea al código original (línea encargada de identificar la IP del ordenador en que esta conectado el Pan&Tilt), y "linkando" las librerías modificadas en lugar de las originales al compilar el programa.

- Todas las funciones y opciones de las librerías originales permanecerían inalteradas, obteniendo un mayor control sobre la posición del Pan&Tilt.

5.2.3 DECISIÓN

Después de implementar ambas soluciones y de comparar los pros y contras de ambas, se constató que aunque la segunda solución era más lenta, la diferencia no compensaba las grandes ventajas que esta aportaba, por lo que se tomó como válida. La diferencia entre la librería original y la modificada es mínima, y esta se encuentra en el fichero W32socket.cpp (W32seria.cpp en el original).

Solucionado el problema de la conexión, se afronta el control mismo del Pan&Tilt y de las diversas posibilidades de las que se dispone.

Las funciones que incorporan las librerías del fabricante permiten diversos modos de controlar el Pan&Tilt. Por velocidad, posición y aceleración, son los tres modos básicos de control de que se dispone, aunque serán particularmente los dos primeros los más necesarios para el desarrollo de la interfaz, como se muestra a continuación:

El control por velocidad permite mover el Pan&Tilt de modo continuo, sin "ir a saltos". Si cuando se pulsa una tecla de control en la interfaz, se diese una orden de posición (aumentar o disminuir una determinada cantidad de grados la orientación de uno de los ejes), aunque se enviase otra inmediatamente, el Pan&Tilt finalizaría la primera deteniéndose antes de ejecutar la siguiente, provocando así una evolución del mismo "a saltos". Por otro lado, esta manera de proceder supondría un gran flujo de información a través del Ethernet, ya que mientras se mantuviese el Pan&Tilt en movimiento, el Navegador debería estar enviando continuamente consignas de posición, produciéndose pérdidas de paquetes de datos y empeorando el tiempo de respuesta.

Para evitar este tipo de problemas se utilizará el control por velocidad. Al pulsar una tecla en la interfaz, el Pan&Tilt comenzará a moverse con una velocidad constante (fijada a priori por el usuario) y no parará a no ser que se le indique lo contrario. De esta manera el movimiento es totalmente fluido hasta que dejamos de pulsar la tecla (porque se pulsa otra, o simplemente se quiere detener el movimiento), momento en que se manda la orden de detenerse. Esta manera de actuar, además de evitar la evolución del Pan&Tilt "a saltos", no supone una sobrecarga de trabajo para el radio-módem, enviando información solo al inicio y final de movimiento, y no durante toda la realización del mismo.

Por otro lado, el control por posición es mucho más adecuado en otros casos. En algunas situaciones, el usuario querrá fijar la orientación del Pan&Tilt sin tener que estar pendiente de la evolución del mismo, simplemente darle una consigna de posición y que el mismo se encargue de orientarse. Es en estas ocasiones donde el control por posición se revela como el más apropiado, al permitir dar las consignas de posición tanto absolutas

como relativas. Una vez dada la orden, el Pan&Tilt comenzará a girar hasta que alcance la orientación deseada, momento en que se detendrá automáticamente.

5.3 CONTROL DEL ROBOT

Como ya se comentó en la descripción software del proyecto (ver capítulo 3, apartado 3.2.2.), Saphira utiliza un sistema de trabajo cliente/servidor que facilita enormemente la tarea de conectar vía Ethernet con el robot, por lo que en este caso no hubo necesidad de diseñar un sistema de comunicación.

Saphira permite básicamente dos modos de control del robot:

- Control directo: Gracias al cual el usuario manda ordenes directas al robot, tales como desplazarse a una posición, girar en un sentido con una velocidad, etc ...

- Control por comportamientos: Permite dotar al robot de "comportamientos", posibilitándolo para cumplir tareas mucho más complejas que una simple traslación o rotación.

5.3.1. CONTROL DIRECTO

La capacidad de Saphira para implementar sistemas de control directo de alto nivel permite crear programas-cliente capaces de conectarse al robot y de intercambiar información, enviarle ordenes de desplazamiento, etc ..., sin necesidad de entrar en el control del robot al más bajo nivel.

El control directo esta dividido en dos "canales", por un lado el control de la traslación, y por otro el de la rotación; teniendo funciones específicas para el control de cada una de ellas.

A continuación se hace un breve resumen de las funciones que Saphira pone a disposición del programador para realizar el control directo del robot:

---

void sfSetVelocity (int vel)
void sfSetRVelocity (int rvel)

Fijan la velocidad de traslación y de rotación respectivamente. Los valores de la velocidad de traslación deben estar en mm/seg y los de la rotacional en grados/seg. Estas son las funciones que se encargan del control del robot por velocidad. Si **vel** o **rvel** son diferentes de cero, el robot comienza a trasladarse o rotar respectivamente.

void sfSetHeading (int head)
void sfSetDHeading (int dhead)

La primera función establece orientación absoluta del robot en grados (de 0 a 359), mientras que la segunda establece un incremento o decremento de **dhead** grados con respecto a la orientación actual del robot. Controlan la orientación del robot por posición absoluta o relativa, respectivamente.

void sfSetPosition (int dist)
void sfSetMaxVelocity (int vel)

En el primer caso se produce una traslación de **dist** mm, hacia delante si es positivo y hacia atrás si es negativo. La máxima velocidad obtenida durante la traslación viene determinada por la segunda función.

int sfDonePosition (int dist)
int sfDoneHeading (int ang)

Ambas funciones comprueban que se haya cumplido una orden de dirección previamente realizada. La primera función verifica si se ha realizado una traslación, y el argumento especifica a que distancia del objetivo tiene que estar el robot para que esta se considere completada. La segunda efectúa la misma función pero en este caso verifica una rotación.

float sfTargetVel (void)
float sfTargetHead (void)

En este caso, la primera función nos informa de cual es la velocidad actual del robot, medida en mm/seg; mientras que la segunda nos notifica la orientación del robot, medida en grados.

Como sucedía en el caso del Pan&Tilt, para el guiado del robot se han utilizado rutinas de control por velocidad para evitar el fenómeno de trabado durante el desplazamiento (ver apartado 5.2.3 Conclusión). No se va explicar de nuevo la conveniencia de utilizar consignas de velocidad en el control manual del robot, sin embargo lo que sí se va a incluir es el algoritmo utilizado a tal fin (página siguiente, *Figura 5.2.*). Este algoritmo podría aplicarse fácilmente al control del Pan&Tilt, substituyendo las ordenes de avanzar, retroceder, girar derecha y girar izquierda, por girar Tilt abajo, girar Tilt arriba, girar Pan derecha y girar Pan izquierda, respectivamente.

```
Comprobar estado de los controles de movimiento( );
Si ha cambiado el estado del control  avanzar Entonces
{
        Mirar estado actual del control avanzar:
        {
        Caso Activado:
                Mandamos orden al robot de comenzar a avanzar a velocidad constante
        Caso Desactivado:
                Mandamos orden al robot de detener solo la traslación
        }
}
Si ha cambiado el estado del control retroceder Entonces
{
        Mirar estado actual del control retroceder:
        {
        Caso Activado:
                Mandamos orden al robot de comenzar a retroceder a velocidad constante
        Caso Desactivado:
                Mandamos orden al robot de detener solo la traslación
        }
}
Si ha cambiado el estado del control girar derecha Entonces
{
        Mirar estado actual del control girar derecha:
        {
        Caso Activado:
                Mandamos orden al robot de comenzar a girar sobre si mismo en el
                sentido de las agujas del reloj a velocidad constante
        Caso Desactivado:
                Mandamos orden al robot de detener solo la rotación
        }
}
Si ha cambiado el estado del control girar izquierda Entonces
{
        Mirar estado actual del control girar izquierda:
        {
        Caso Activado:
                Mandamos orden al robot de comenzar a girar sobre si mismo en
                sentido opuesto al de las agujas del reloj a velocidad constante
        Caso Desactivado:
                Mandamos orden al robot de detener solo la rotación
        }
}
Obtenemos posicion y orientacion actuales del robot ( );
Repintar escena en funcion de los valores que obtenemos de la funcion anterior ( );
```

Figura 5.2: Algoritmo de control directo del robot

5.3.2. CONTROL POR COMPORTAMIENTOS

El control directo es apropiado cuando se trata de mover el robot mediante simples secuencias de acciones. Sin embargo, en ciertos casos, el robot debe de cumplir tareas más complejas, así como mantener varios objetivos al mismo tiempo. Es en estos casos, como por ejemplo la traslación del robot de un punto a otro, no necesariamente unidos por una recta, a una velocidad constante y evitando los posibles obstáculos, cuando el control por comportamiento se hace prácticamente imprescindible.

Una de las posibles aproximaciones a la resolución de problemas de control complejo, es descomponer la tarea a realizar en pequeñas acciones que cumplan objetivos concretos. De esta manera, cada pequeña acción, junto con su objetivo asociado, recibe el nombre de *comportamiento*. Así, en el ejemplo anteriormente mencionado de trasladar el robot de un punto a otro, se descompondría la tarea en 3 comportamientos:

- Alcanzar el punto de destino
- Mantener una velocidad de traslación constante
- Evitar posibles obstáculos.

Todos y cada uno de estos comportamientos intenta llevar a cabo su objetivo al mismo tiempo que los demás. ¿Como decide Saphira que comportamiento tiene que tomar el control del robot en cada momento?. La respuesta es *Control por lógica Difusa*.

Saphira tiene unos cuantos comportamientos predefinidos, tales como:

sfConstantVelocity	Pone el robot en movimiento hacia delante, a una velocidad constante
sfStop	Detiene el robot cualquier traslación o rotación del robot
sfKeepOff	Separa suavemente el robot de los obstáculos, girándolo si es necesario.
sfAvoidCollision	Muy similar a sfKeepOff pero mucho más configurable
sfStopCollision	Detiene el robot suavemente cuando encuentra un obstáculo. A difererencia de sfKeepOff o sfAvoidClollision, no gira el robot.
sfGoToPos	Desplaza el robot hasta una determinada distancia de un punto fijado por el usuario.

sfAttendAtPos	Muy similar a sfGoToPos
SfFollow	Este comportamiento hace que el robot se dirija hacia un punto siguiendo una recta definida por el usuario
sfFollowCorridor	Mediante este comportamiento obligamos a que el robot se desplace a lo largo de un pasillo predefinido por el usuario, hasta un punto de destino
SfFollowDoor	Instamos al robot a que entre por una puerta. No hay que olvidar que Saphira permite la definición de punto orientados, pasillos, puertas, etc ...
SfTurnTo	Mediante este comportamiento el robot gira para orientarse hacia un punto definido por el usuario

Estos comportamientos pueden ser invocados desde un código en C o desde un fichero de actividades.

Para hacer accesibles a la interfaz todas estas opciones, se decidió incorporar en ella los comportamientos que se consideraron imprescindibles, que son:

- **Velocidad constante (sfConstantVelocity):** Permite desplazar el robot hacia delante a una velocidad constante sin necesidad de guiarlo manualmente. Sus parámetros son:
  - Velocidad de traslación: Velocidad a la que se desea que el robot se desplace, expresada en milímetros por segundo
- **Evitar Colisión (sfAvoidCollision):** Este comportamiento activado, en conjunción con el comportamiento de Velocidad constante, permite dejar al robot solo para que se mueva con total libertad sin miedo a que colisione con objetos del entorno. Sus parámetros son:
  - Sensibilidad a los obstáculos frontales: Define la distancia a la que el robot considera que los obstáculos frontales están demasiado cerca. Su valor oscila entre 0.5 (insensible) y 3 (muy sensible).
  - Sensibilidad a los obstáculos laterales: Define la distancia a la que el robot considera que los obstáculos laterales están demasiado cerca. Su valor oscila entre 0.5 (insensible) y 3 (muy sensible).

- Ganancia en el giro: Controla la velocidad de giro del robot cuando considera que hay un obstáculo demasiado cerca. Su valor oscila entre 4.0 (giro lento) y 10.0 (giro rápido).

- Radio de seguridad: Define la esfera de seguridad que rodea al robot. Debe de estar expresado en milímetros.

- **Mantener Distancias (sfKeepOff):** Si durante el funcionamiento del robot, ya sea por guiado manual o mientras se traslada de un punto a otro (pathplanning), se mantiene este comportamiento activado, se evita el riesgo de que, por un "error de cálculo", el robot colisione con algún obstáculo. Sus parámetros son:

  - Velocidad de precaución: El robot establece esta velocidad cuando obstáculos distantes son detectados. El valor debe estar expresado en milímetros por segundo.

  - Sensibilidad a los obstáculos: Tanto a los frontales como laterales. Este parámetro cumple la misma función que sus homónimos en sfAvoidCollision. Su valor puede variar entre 0.2 (insensible) y 2.0 (muy sensible).

- **Detener Robot (sfStop):** Prácticamente imprescindible, de hecho es el único comportamiento al que se puede acceder directamente desde la botonera del Navegador, sin necesidad de ir al menú de comportamientos. Activándolo se detiene cualquier actividad que el robot estuviese realizando en ese momento; incluso el guiado manual queda parcialmente deshabilitado, ya que será imposible mover de sitio el robot, tan solo será posible rotarlo. Este comportamiento no tiene parámetros.

En un principio la intención era poder definir cada uno de los parámetros de los comportamientos desde la interfaz de control. Estos serían cargados con los parámetros definidos por el usuario, y descargados cuando este quisiera detenerlos o simplemente variar los parámetros. Pero hubo que cambiar la estrategia a seguir para llevar la idea a la práctica.

Después de muchas pruebas (incluso con la misma interfaz que el propio Saphira incorpora para el control del robot) se constato que Saphira se vuelve inestable al cargar y descargar los comportamientos en memoria. Al cargar un comportamiento se desactivaban otros ya cargados, o viceversa, etc... lo que provocaba una total impredictibilidad en su funcionamiento. Sin embargo se comprobó que si se cargaban todos los comportamientos a la vez, sí que se conseguía que todos y cada uno de ellos siguieran los parámetros que se les indicaban. Por tanto la nueva estrategia a seguir era clara.

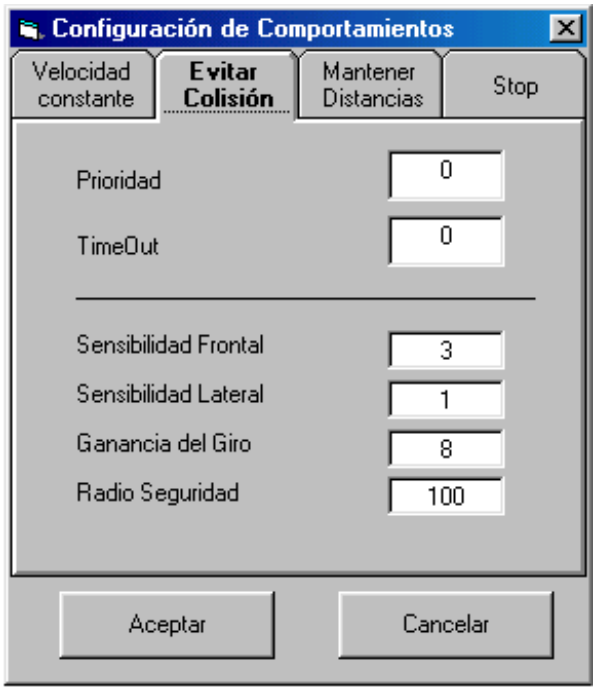


Figura 5.3: Imagen del cuadro de diálogo de nuestra interficie, que permite configurar los parámetros de los comportamientos predefinidos por Saphira

Cuando la aplicación conecta con el robot, esta carga todos los comportamientos pero en estado latente (pausados). En el momento que se desee activar uno de los comportamientos, cambiar algunos de sus parámetros, desactivar comportamientos que ya estuviesen activos, etc... lo que se hace es descargar de memoria todos los comportamientos, no solo el que haya sufrido la modificación sino todos. Inmediatamente después se vuelven a cargar de nuevo todos los comportamientos con los parámetros actuales introducidos por el usuario. El proceso de descarga y carga de los comportamientos es muy rápido, por lo que en el caso de que el robot se este desplazando, solo apreciaremos una brevísima pausa en su movimiento y el consiguiente cambio en su manera de comportarse (en función de los nuevos parámetros establecidos por el usuario).

Ahora el usuario ya cuenta con un juego básico de comportamientos que le permiten controlar el robot manualmente sin preocuparse demasiado de colisionar con un obstáculo, he incluso detener cualquier tipo de traslación en caso de emergencia. Pero, como se comenta al principio de este capítulo, la posibilidad de implementar nuestros propios comportamientos en C y , sobretodo, la posibilidad de definir actividades, abren un abanico de posibilidades mucho más amplio que el de los comportamientos predefinidos por Saphira. Tareas como localizar un objeto del entorno y girar alrededor de él para obtener información de su contorno, no son tan simples de realizar y precisan la definición de un fichero de actividad, donde se carguen las DLL's necesarias y se ejecuten los comportamientos apropiados. No se va a entrar a explicar en detalle como se construyen las actividades y sus comportamientos, puesto que no es el objeto de este proyecto, pero es evidente la conveniencia de dar la posibilidad al usuario de cargar actividades desde la interfaz.

Con este Objetivo se llama a la función de Saphira `sfLoadEvalFile` cuyo único parámetro es el nombre de un fichero. Gracias a esta función es posible importar a la aplicación una actividad desarrollada por cualquier persona, con todo lo que ello implica (desarrollo de nuevos comportamientos, etc...). Pero, ¿cómo se puede detener la actividad

que acabamos de cargar?. En el caso de los comportamientos la respuesta era sencilla ya que gracias a las funciones:

- `sfInterruptTask(char *iname)`
- `sfResumeTask(char *iname)`
- `sfRemoveTask(char *iname)`

Estas funciones dan la posibilidad de pausar un comportamiento, reasumir su funcionamiento, e incluso descargarlo de memoria respectivamente, solo con hacer referencia al Iname(nombre de la instancia) con el que Saphira identifica internamente los comportamientos. Este nombre de instancia lo define el programador cuando carga en memoria el comportamiento, pero en el caso de los ficheros de actividades, tal y como hemos dicho, el único parámetro que le pasamos a la función `sfLoadEvalFile` es el nombre del fichero. ¿Cómo identifica entonces Saphira esta actividad?. La respuesta no es evidente pero sí bastante simple. Puesto que no se puede definir el nombre de la Instancia, Saphira le asigna como tal el nombre del fichero que le cargamos pero sin la extensión. Es decir, si se intenta cargar una actividad concreta que se encuentra en la unidad H:\, se ejecutará la orden:

```
SfLoadEvalFile ("h:\sfTrackingObject.act")
```

Al ejecutar esta orden, Saphira carga la actividad y le asigna automáticamente como nombre de instancia sfTrackinObject. Por tanto ya es posible pausarla, reasumirla o descargarla con las mismas funciones que se utilizaban en el caso de los comportamientos.

6 PATH PLANNING

6.1. INTRODUCCIÓN

La generación de trayectorias o "path planning", tal y como aparece en la literatura inglesa, tiene como fin la confección de un camino libre de colisiones para un robot móvil.

A la hora de plantear el problema de la generación de trayectorias, son dos los principales aspectos a tener en cuenta:

- La naturaleza del robot.
- El tipo de entorno

Naturaleza del robot: A priori, las características del robot y sus limitaciones son cuestiones a tener muy en cuenta antes del desarrollo de un algoritmo generador de trayectorias. Características tales como los grados de libertad del robot, restricciones tanto en los giros como los desplazamientos, dimensiones, radio de giro, etc ...

Principalmente podemos clasificar los robots en dos grandes grupos:

- Robots holonómicos: Poseen tantos grados de libertad independientes como el número máximo de grados de libertad posibles que admite la dimensión del entorno. Esto quiere decir que en un entorno 2D, el número máximo de grados de libertad son 3: la coordenada X, la coordenada Y, y la orientación.

- Robots no holonómicos: En este caso, los robots presentan restricciones en cuanto a su capacidad de desplazamiento. Poseen menos grados de libertad que los necesarios para completar un tarea, o simplemente los grados de libertad que presentan no son independientes (robots tipo "car-like").

Tipo de entorno: El otro dato a tener presente es el entorno en el que el robot va a realizar sus evoluciones. Podemos agrupar los entornos en:

- Interiores: Como su propio nombre indica, los interiores son entornos cerrados. Estos medios se caracterizan porque los objetos que pueden encontrarse en ellos son fácilmente representables mediante figuras geométricas, la iluminación esta muy controlada, la disposición de los obstáculos y su naturaleza es bastante previsible, etc ... Todas estas características facilitan enormemente la detección de características en los algoritmos de estéreo-visión, correspondencia, autoposicionamiento, etc ... Alguno ejemplos de interiores serian las habitaciones de cualquier edificio o fábrica.

- Exteriores: Los entornos externos dificultan enormemente el desarrollo de algoritmos de "path planning" debido a la aleatoriedad del mismo. Todos los algoritmos de autoposicionamiento, estéreo-visión, etc ... se ven fuertemente afectados por lo imprevisible del entorno. Los entornos exteriores pueden ir desde una superficie extraterrestre (tomemos como ejemplo la superficie marciana en el caso del "PathFinder"), hasta un entorno algo más controlado como las calles de una ciudad.

En cuanto al caso concreto que nos ocupa, se trata de un robot no holonómico en un entorno interior.

6.2. SOLUCIÓN AL PROBLEMA DEL "PATH PLANNING"

La solución adoptada esta basada en un proyecto final de carrera ya desarrollado (GaJ 99), adaptándolo a la situación concreta de este proyecto, realizando las modificaciones oportunas.

El método empleado para la implementación del algoritmo es el probabilístico, que consta principalmente de dos fases bien diferenciadas:

- Construcción del mapa de trayectorias
- Búsqueda del camino más corto.

6.2.1. CONSTRUCCIÓN DEL MAPA DE TRAYECTORIAS

En esta fase del algoritmo se pretende generar todo un mallado de posibles trayectorias que unan los puntos de salida y llegada, sin que estas intercepten ningún obstáculo, cumpliendo además con las restricciones propias del robot.

Para ello, en un primer paso, se lanza una determinada cantidad de puntos aleatorios sobre la imagen que posteriormente intentaran ser unidos entre sí. Es aquí donde se realiza la primera modificación con respecto al algoritmo original.

Puesto que la generación de nodos del mallado es totalmente aleatoria, la probabilidad de que alguno de estos nodos coincida con el punto de destino elegido por el usuario, y con el punto en el que esta el robot en ese momento (puntos de destino y de partida respectivamente) es mínima. El programa original solucionaba este problema seleccionando de entre los nodos generados, los más cercanos (ver *Figura 6.1*), pero es evidente que en una aplicación real, tanto el punto de destino como el de partida (sobre todo este último) es prácticamente imprescindible que sean los reales.

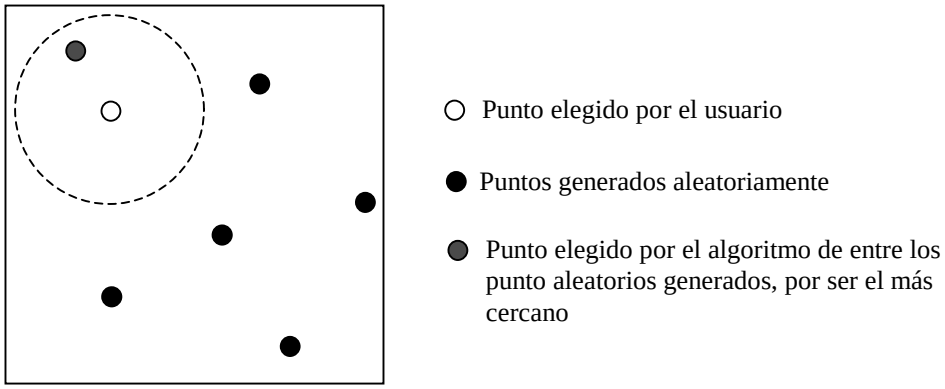


Figura 6.1: Ejemplo de como seleccionaba el algoritmo original, los puntos de destino o de partida

La solución a este problema es forzar la introducción de los puntos deseados (tanto la posición actual del robot como el punto de destino) dentro de la estructura de datos que almacena la información de la distribución de puntos aleatorios. Simplemente debe tenerse en cuenta que, a continuación, los puntos a generar son N-2 puntos aleatorios, siendo N el número total de puntos a generar.

Como ya se menciona al principio de este mismo apartado, para generar el mallado de posibles trayectorias deben tratar de unirse los puntos generados al azar. Sin embargo, antes de unir dos configuraciones aleatorias el algoritmo debe asegurarse que ambas, así como el camino que las une, cumplen las restricciones del robot. En este caso concreto se hace la suposición que se trata de un robot holonómico, por lo que la única restricción que debe de cumplirse es que el robot pueda pasar de una configuración a otra sin que la superficie que comprende el radio de seguridad entre en contacto con ningún obstáculo del recorrido.

En la *Figura6.2.* (página siguiente), se presenta un ejemplo de conexión imposible entre dos configuraciones. Como se puede observar, la línea recta que une ambos puntos no intersecta con ningún obstáculo, de hecho, es posible que ni siquiera el robot colisionase con el objeto, pero el volumen de seguridad (definido por el usuario

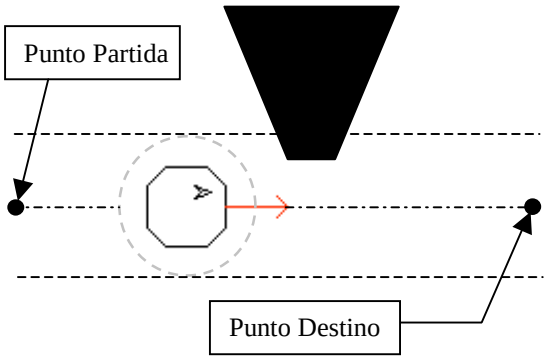


Figura 6.2: Ejemplo de como dos puntos aleatorios que intentan ser unidos entre sí incumplen la restricciones del robot de la imagen. En este ejemplo los puntos de partida y destino son 2 configuraciones aleatorias cualesquiera

mediante un radio de seguridad, y representado en la imagen por la línea punteada que rodea el robot) sí, por lo que jamás deberían de unirse estos nodos.

Es en esta parte del algoritmo, en la rutina que se encarga de verificar la posibilidad de conexión entre dos configuraciones aleatorias, donde se han tenido que hacer las modificaciones más profundas.

En el algoritmo original se comprobaban los puntos situados en el perímetro de un disco de radio de seguridad R (suministrado por el usuario), distribuidos a intervalos regulares de 60 grados, a lo largo de todo el trayecto que uniría las dos configuraciones (ver *Figura6.3.*). Aunque en principio se puede pensar que se trata de una solución suficientemente buena, determinadas disposiciones de los puntos pueden provocar autenticas zonas muertas que no serían revisadas a la hora de hacer las conexiones. La peor situación que podría darse sería la de la *Figura6.4*.

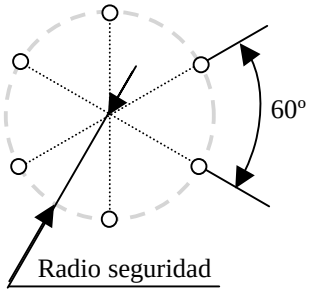


Figura 6.3: distribución de los puntos donde se verifica la existencia de obstáculos

Como se puede observar, si los puntos que tratamos de unir forman una trayectoria de 30º con respecto

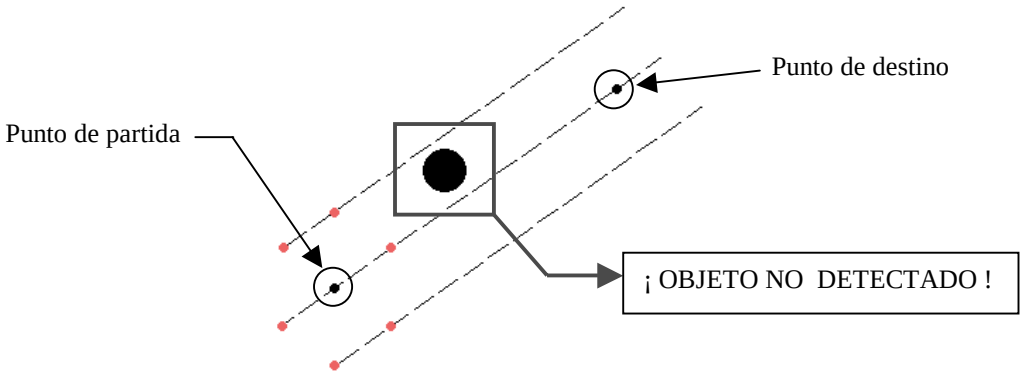


Figura 6.4: Los dos puntos que se tratan de unir están alineados de tal manera que los puntos del perímetro de seguridad también quedan alineados dos a dos. Esto provoca que en lugar de tener 6 trayectorias en las que se verifica la existencia de obstáculos (representadas en el gráfico por líneas punteadas) tengamos tan solo 3, dejando un gran espacio entre ellas. Cuanto más grande el radio de seguridad más espacio habría entre líneas, pudiendo llegar a albergar objetos peligrosamente grandes entre ellas.

a la horizontal, los puntos pertenecientes al contorno del disco de seguridad, encargados de verificar la existencia de objetos, quedan alineados. Esto provoca que un objeto pueda pasar totalmente desapercibido entre las líneas de puntos que se revisan. Con un simple cálculo se puede comprobar que con un radio de seguridad de 25 cm, podría no ser detectado un objeto de hasta 20 cm de diámetro.

Un inconveniente adicional del método es que la velocidad de este es directamente proporcional a la distancia que une las dos configuraciones, con lo cual aunque dos puntos muy cercanos serian estudiados muy rápido, dos puntos al doble de distancia tardarían el doble en ser analizados.

Se imponía la necesidad de encontrar un método independiente a la resolución del mapa, así como robusto a cualquier situación y objeto. Por este motivo se ha diseñado un método basado en operaciones morfológicas sobre imágenes, el algoritmo del cual sigue los pasos que se describen a continuación:

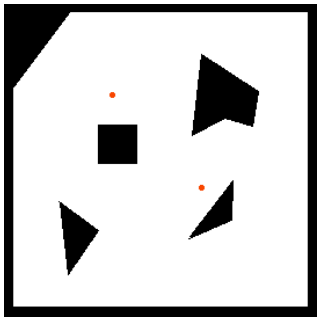


Figura 6.5.

Supóngase que el mapa 2D de la zona sobre la que el robot esta realizando sus evoluciones es el de la *Figura 6.5*, y que durante el proceso de creación de la malla de posibles caminos, el algoritmo quiere saber si el robot (más concretamente el disco de seguridad que lo rodea), podría desplazarse de uno de los puntos señalados en el mapa al otro.

El proceso que seguiría el algoritmo que se ha desarrollado es:

En primer lugar, se reserva espacio en memoria para un "buffer" en el que se va a generar una nueva imagen. Esta imagen debe tener el fondo blanco, porque va a pintarse en negro la superficie sobre la que se desplazaría el disco de seguridad del robot, al ir de uno de los puntos al otro (ver *Figura 6.6*). Esta superficie es pintada teniendo en cuenta la escala del mapa que estamos analizando.

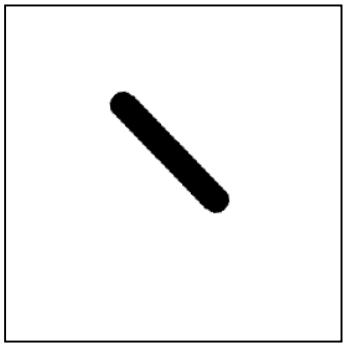


Figura 6.6.

Una vez obtenida esta imagen (todo el proceso es invisible para el usuario), se procede a realizar una suma boleana (OR) entre la imagen del mapa y la que se hemos generado sintéticamente. Este proceso lo realiza una función de las librerías MIL, y consiste en una comparación píxel a píxel entre las dos imágenes según la cual, solo si los píxeles de ambas son negros, la imagen que genera tiene el píxel correspondiente negro.

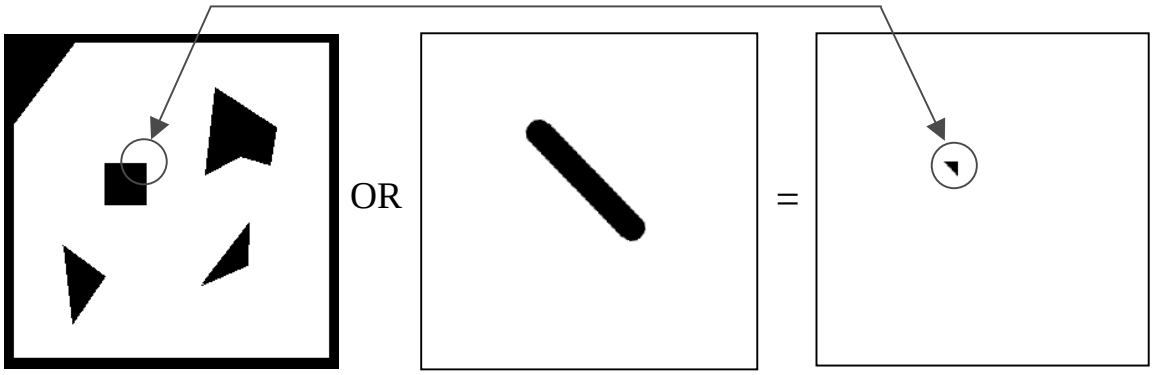


Figura 6.7: Después de hacer el OR entre las dos imágenes, podemos observar como en la imagen resultante aparece una esquina en negro que corresponde con la esquina marcada en el plano.

En la imagen resultado (ver *Figura 6.7*), como consecuencia de la aplicación del OR (ver *Figura6.8* para tabla de verdad), aparece una esquina en negro. Esto significa que durante la traslación del robot de uno de los puntos al otro, su disco de seguridad si que entraría en contacto con uno de los obstáculos del mapa. El último paso que realizaría el algoritmo sería precisamente este, verificar si en la imagen resultado existe algún píxel negro y, en caso afirmativo, dar como NO válida la conexión de los puntos.

Imagen A	Imagen B	A OR B
0	0	0
0	1	1
1	0	1
1	1	1

Figura 6.8: Tabla de verdad de una operación OR. Se ha que tener en cuenta que para nosotros el blanco es un 1, y el negro un 0

Este método es mucho más robusto que el original ya que cualquier objeto que aparezca representado en el mapa, por pequeño que sea, si coincide con la superficie de desplazamiento del robot, dejara una "marca" negra en la imagen resultante. Aun cuando tan solo sea un píxel lo que aparezca en la imagen resultante, se desecha la conexión de los puntos.

Hay que mencionar que el algoritmo de generación de mallas dispone de una colección de parámetros configurables entre los cuales destacan: el método de conexión entre nodos, gracias al cual disponemos de dos opciones, o generamos primero todos los nodos y luego tratamos de unirlos (pregeneración de nodos), o intentamos unir los nodos a medida que los vamos generando (conexión punto por punto); la distancia mínima y máxima entre nodos para intentar una conexión; el número máximo de conexiones por nodo; o el número máximo de vecinos a los cuales conectar el nodo actual, etc...

En este momento ya es posible generar un mallado de posibles caminos, pero antes se debería de tener en cuenta una consideración más.

Ahora que ya se ha tratado el modo en que se realiza la conexión entre nodos y las restricciones que ello conlleva, es lógico pensar que en determinadas zonas (donde el disco de seguridad del robot difícilmente permitiría una conexión válida), el método de generación de nodos para el mallado es deficiente. Para solucionar este problema se introdujo la idea de las ventanas de resolución. La función de estas ventanas de resolución es la de obligar al generador de puntos aleatorios a que lance un determinado porcentaje de estos puntos dentro de un recinto definido por el usuario. El algoritmo original permitía definir estas ventanas de resolución, así como su posición en el mapa y el porcentaje de puntos que queríamos que se definiesen en su interior. Lo que no permitía era definir su forma, es decir, la ventana de resolución era una cuadrado de dimensiones fijas independientemente de la resolución del mapa, lo que reducía su utilidad en mapas con un zoom muy grande (ver *Figura 6.8*).

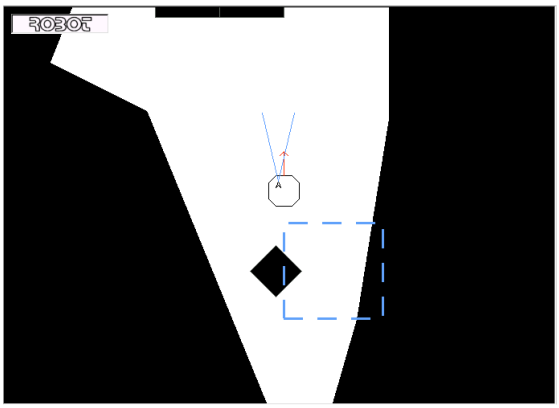


Figura 6.8A

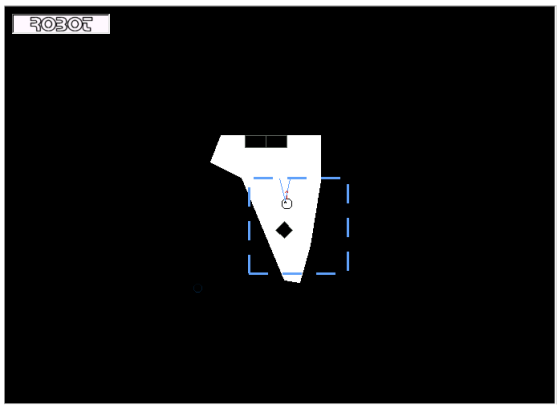


Figura 6.8B

Figura 6.8: Como se puede apreciar en la *Figura 6.8A*, la ventana de resolución (definida por el cuadrado punteado) nos permite ampliar la resolución de la malla en el punto conflictivo que se encuentra entre el objeto y la pared. Sin embargo, en caso de que tuviésemos un Zoom muy grande (para este mapa no tiene mucho sentido pero es un ejemplo), en la *Figura 6.8B* vemos que al no poder redefinir el tamaño de la ventana de resolución se hace virtualmente imposible concretar la zona conflictiva, reduciendo notablemente la utilidad de esta.

Para solucionar este problema se mejoró la interactividad del usurario en la definición de las ventanas de resolución. Ahora el usuario puede definir su ventana de resolución pulsando el botón izquierdo del ratón en conjunción con la tecla CTRL del teclado, y arrastrándolo de manera que quedan definidas las esquinas de una caja. Pero lo mas importante es que esta ventana de resolución se traslada, rota y se escala junto con el mapa, solucionando el problema anteriormente mencionado (ver *Figura 6.9* página siguiente).

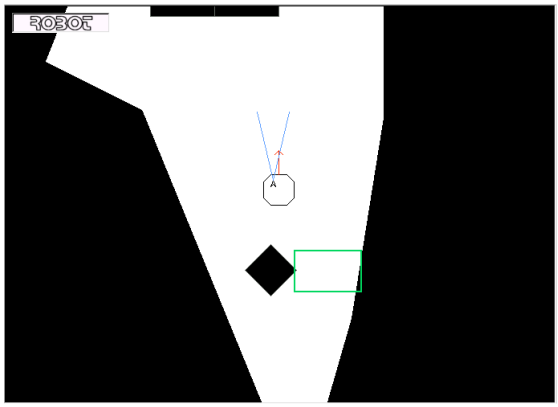


Figura 6.9A

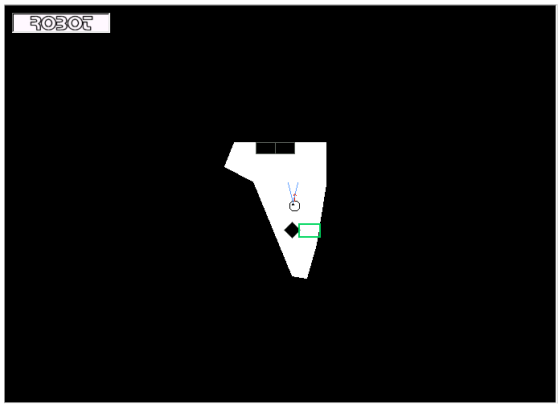


Figura 6.9B

Figura 6.9: En la *Figura 6.9A* podemos apreciar como la ventana de resolución (definida en este caso por un recuadro verde) no tiene un tamaño fijado, a fin de poder definir la zona lo más concretamente posible. En la *Figura 6.*9B vemos como la ventana de resolución se ha escalado junto con el mapa.

Por ejemplo:

Supóngase que tenemos un mapa como el de la *Figura 6.10*, y que se pretende que el robot se desplace al punto señalado en azul.

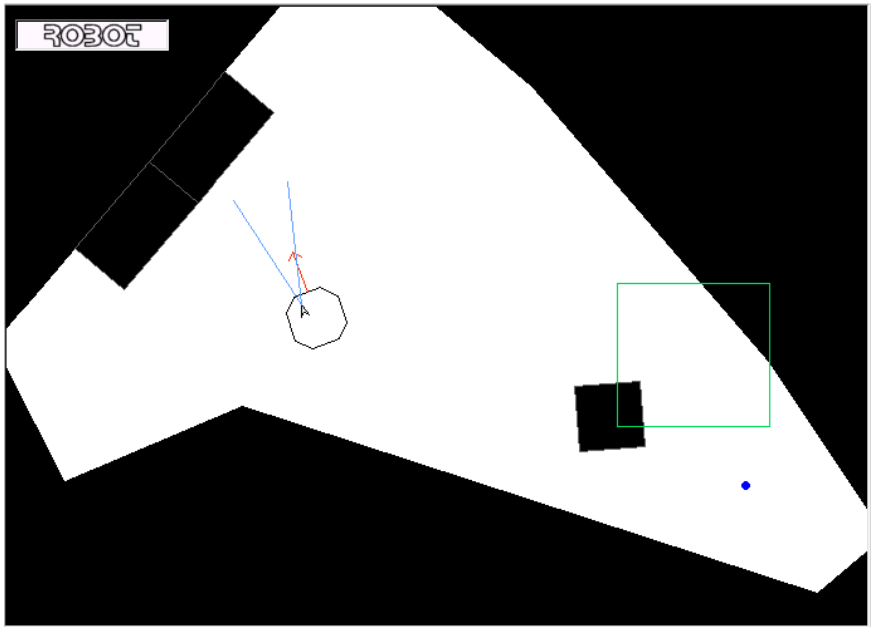


Figura 6.10: En la figura el punto de destino esta señalado con un punto azul, el cual no se vería durante el uso normal del Navegador. En este caso esta señalado a fin de que el lector pueda saber cual es el punto de destino del robot. Como puede observarse, hemos definido una ventana de resolución en previsión del posible conflicto que puede tener lugar en la zona encuadrada

Se definen los parámetros de la generación del mallado o "Roadmap" y se obtiene (ver *Figura 6.11*):

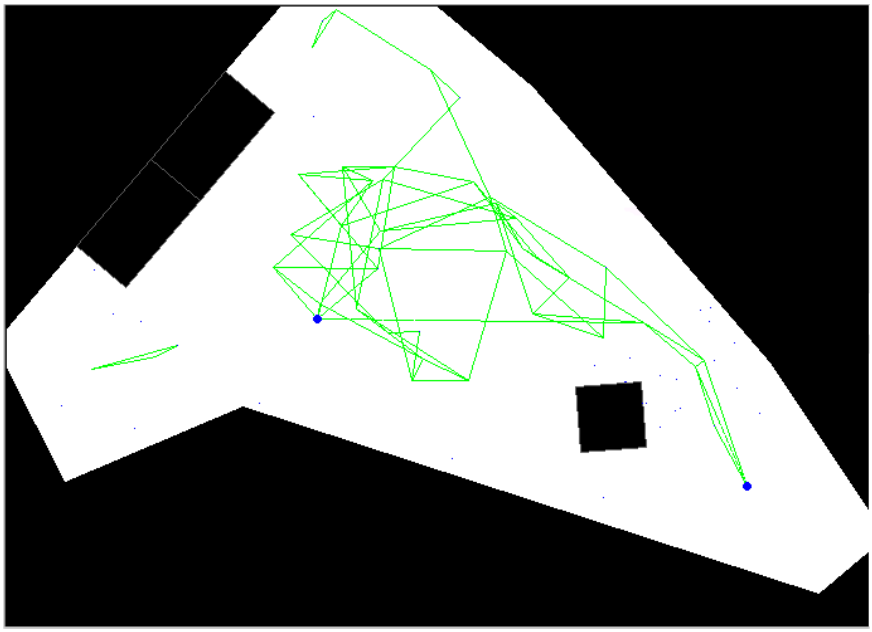


Figura 6.11: Este es el mallado (o "Roadmap") obtenido con los siguientes parámetros:

Conexión punto por punto
Número de nodos generados = 70
Distancia máxima entre nodos= 1 metro
Distancia Mínima entre nodos = 2 milímetros
K vecinos = 6
Número máximo de conexiones por nodo = 4
Porcentaje puntos en zona alta resolución = 30%

6.2.2. BÚSQUEDA DEL CAMINO MÁS CORTO

6.2.2.1. PRIMERA APROXIMACIÓN A LA SOLUCIÓN

Una vez generada la malla de puntos, el siguiente paso es determinar las posibles trayectorias que unen el punto de partida y el de destino, para quedarnos con la más corta.

El método empleado básicamente consiste en explorar todos los caminos posibles que existen en el mallado partiendo del punto inicial, para quedarnos con los que conecten con el punto final, y de entre estos los que minimizan la distancia. En la *Figura6.12* (ver página siguiente), tenemos el algoritmo utilizado para realizar la búsqueda.

```
Nodo=Salida
Final=falso
Mientras final = falso hacer
{
        Si salta=cierto Entonces
        {
                Retroceder un nivel ( )
                Salta=falso
                Si  Nivel_actual > 0 Entonces
                {
                        Seleccionar nodo actual ( )
                }
                Sino
                {
                        Final=cierto
                }
        }
        Ok=falso
        Expandir Nodo y seleccionar Siguiente Nodo ( )
        Mientras Salta = falso Y Ok = falso Hacer
        {
                Si Siguiente Nodo ya ha sido visitado Y Nodo no explotado totalmente Entonces
                {
                        Expandir Nodo y Seleccionar Siguiente Nodo ( )
                        Si Nodo = Llegada Entonces
                        {
                                Salta = cierto
                                Llamar Otro Path ( )
                        }
                }
                Sino
                {
                        Si Nodo explotado totalmente Entonces
                        {
                                Salta = cierto
                        }
                        Si Salta = falso Entonces
                        {
                                Ok=cierto
                                Nodo=Siguiente Nodo ( )
                        }
                }
        }
}
```

Figura 6.12: Algoritmo de búsqueda. Para una explicación más detallada acerca de la teoría de este algoritmo, me remito al Proyecto Final de Carrera *"Planificació de trajectòries de dues dimensions per a robots mòbils"* por Joaquim Galceran Capeta**.**

Por razones obvias, el tiempo de cálculo necesario para realizar la búsqueda del camino más corto, crece exponencialmente con la cantidad de nodos que conforman el mallado. Esto provoca que en algunas ocasiones el tiempo de cálculo necesario para conseguir un camino (siempre dependiendo de la velocidad de cálculo de la máquina donde se ejecuta el programa) sea excesivo. Para solucionar este problema, es posible limitar el tiempo de búsqueda de tres maneras diferentes:

- **Número máximo de caminos generados:** Cuando se esta realizando la búsqueda del camino más corto, cada vez que se encuentra un camino que une los puntos de partida y de destino, se calcula su longitud. Si esta longitud es inferior a la del último camino encontrado, el nuevo camino pasa a ser el elegido, en caso contrario, no se realizan cambios. Mediante este parámetro se establece la cantidad máxima de caminos "buenos" (que unen el punto de partida con el de destino) que se pretende que encuentre el algoritmo. Si se fija este valor a 1, al primer camino que una los puntos de partida y destino entre si, el algoritmo se detendrá. Si se fija el parámetro a N, el algoritmo no se detendrá hasta que encuentre N caminos correctos, devolviendo el más corto de ellos, o no sea capaz de encontrar más caminos. Como se verá más adelante, en ocasiones el resultado es más rápido e igual de bueno si se acepta el primer camino encontrado, que si se espera a que el algoritmo determine todos los caminos posibles para obtener el más corto.

- **Número máximo de iteraciones realizadas:** Con este valor se limita directamente el número de iteraciones por nodo que se realizan antes de dejar de buscar caminos. Es recomendable no tocar este parámetro a fin de limitar los tiempos de búsqueda. Lo mejor es fijarlo a un valor la suficientemente alto como para que jamás se detenga el algoritmo de búsqueda por este motivo.

- **Tiempo máximo de búsqueda:** Como su propio nombre indica, mediante este valor se limita el tiempo de que dispone el algoritmo de búsqueda para arrojar algún resultado.

Si se realiza el "Backtracking" del ejemplo del apartado anterior (*Figura 6.11*), obtenemos como resultado el camino de la *Figura 6.13*.

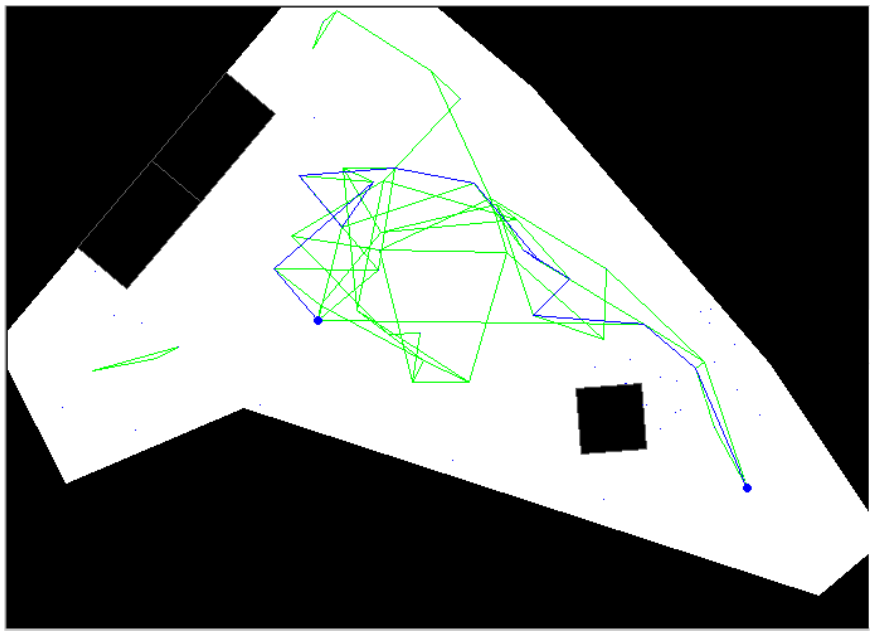


Figura 6.13: Resultado del algoritmo de "Backtracking", limitando el número máximo de caminos encontrados a 10 y el número de iteraciones a 100000. Como puede observarse a simple vista, el algoritmo se detuvo antes de encontrar el camino más corto, cosa que, como veremos a continuación, no tiene porqué tener excesiva importancia.

6.2.2.2. ELIMINACIÓN DE NODOS INNCESARIOS EN LA TRAYECTORIA ESCOGIDA

Como ya se podía observar en el apartado anterior, los caminos obtenidos durante el proceso de búsqueda a partir del mapa de trayectorias son, en la mayoría de los casos, innecesariamente largos. La mayoría de las veces el camino escogido da vueltas redundantes, se aleja del punto de destino para después volver, etc... (ver *Figura6.13*, donde este hecho se hace más que evidente), debido a una gran cantidad de nodos innecesarios en la trayectoria escogida.

Gracias a un sencillo algoritmo de eliminación de nodos superfluos, es posible transformar una trayectoria del todo inapropiada, en la trayectoria final que se buscaba.

Este algoritmo se basa en coger el primer punto de la trayectoria y los dos que le siguen. A continuación se comprueba el resultado de eliminar el nodo central (el que está en medio de los tres escogidos). ¿El robot podría pasar del primer nodo al tercero sin interceptar ningún obstáculo?, si es así se elimina el segundo nodo, si no se deja. Sea cual sea el resultado de la prueba anterior, se toma el siguiente nodo de la trayectoria y sus dos siguientes para realizar la misma operación, y así sucesivamente hasta el último nodo de la trayectoria. Este procedimiento se ejecutará tantas veces como sea necesario, ya que con

una sola "pasada" no es suficiente para eliminar todos los nodos superfluos (ver *Figura 6.14*).

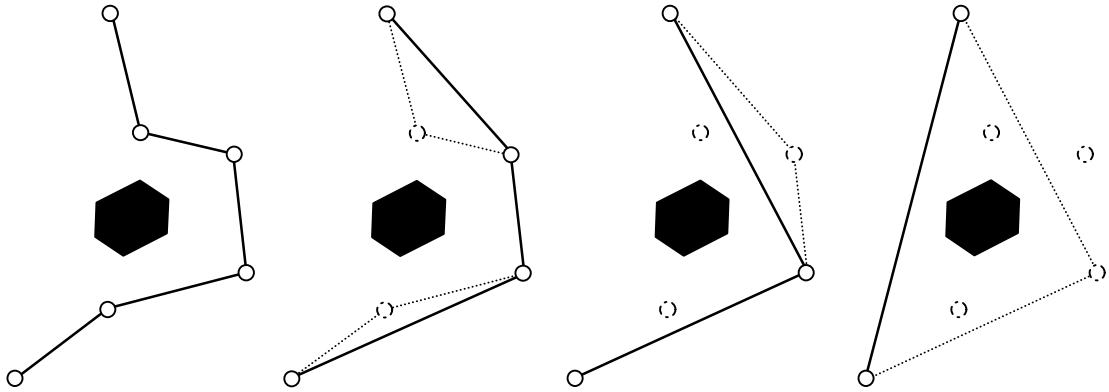


Figura 6.14: En la sucesión de imágenes podemos ver como, ha medida que vamos eliminando nodos innecesarios, estos al igual que sus respectivas conexiones quedan en trazo punteado, mientras que las conexiones válidas tienen un grosor de línea más grande.

Si se utiliza el método de eliminación de nodos innecesarios con el camino resultante del proceso de "Backtracking", se obtiene el siguiente resultado:

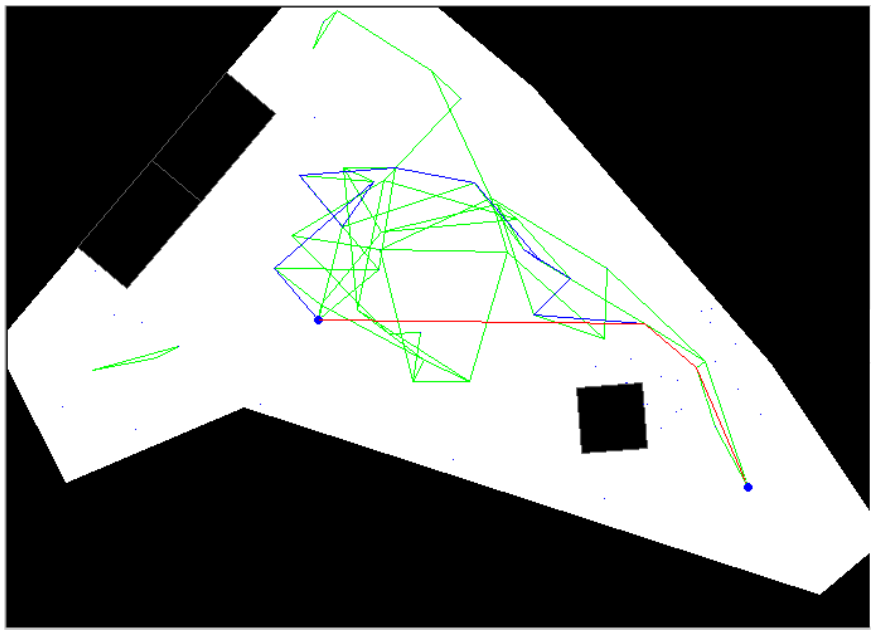


Figura 6.15: El camino rojo es el resultado de eliminar los nodos innecesarios del camino azul

Como se puede observar (*Figura 6.15*), después de eliminar los nodos innecesarios el camino obtenido es notablemente más corto. Debido a la velocidad con que se aplica la reducción del número de nodos, en muchos casos es mucho más rentable hablando en términos de tiempo, que el algoritmo de "Backtracking" no busque todos los caminos posibles sino solo unos cuantos, puesto que en el siguiente paso, el resultado sea posiblemente muy similar y en mucho menos tiempo.

6.2.2.3. ALISAMIENTO DE LA TRAYECTORIA

Hasta el momento, con la trayectoria que ha quedado definida, durante el seguimiento de la misma por parte del robot, este realizaría una serie de traslaciones del punto actual al siguiente de la trayectoria, orientación hacia el siguiente punto de la trayectoria, traslación, etc..., es decir, el robot se traslada durante un tramo rectilíneo, cuando alcanza el final de este tramo, gira sobre si mismo para encarar el siguiente tramo, y así sucesivamente. Este modo de trasladarse es del todo "antinatural". Lo que se pretende es que el robot se traslade de un lugar a otro con un comportamiento "humano". A tal fin debemos de alisar la trayectoria, transformando sus tramos rectilíneos en curvas.

Para conseguir esto hay implementados dos métodos que se pasan a describir a continuación:

- **Alisamiento por radio de giro**

Este método consiste en redondear las esquinas de la trayectoria con el arco de una circunferencia de radio fijo, tangente a los tramos rectilíneos que forman el codo.

- **Alisamiento por Splines**

En este caso el alisamiento se basa en la interpolación mediante Splines, que consiste en una técnica de interpolación polinómica a trozos, muy extendida en gráficos y diseño por computador, que elimina todos los tramos rectilíneos de la trayectoria sustituyéndolos por curvas. Para este proyecto, y como en la mayoría de ocasiones, se utilizan polinomios de tercer orden, ya que son las curvas de menor orden que permiten un punto de inflexión en ellas (paso de cóncava a convexa).

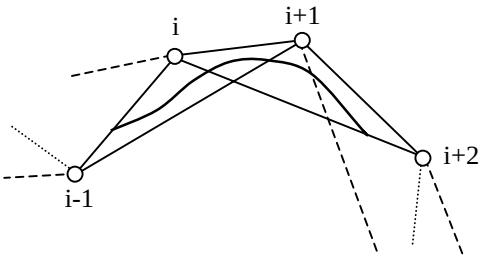


Figura 6.16: vemos como para construir la spline de entre los nodos i,i+1, precisamos de los nodos i-1,i+2, que nos permiten construir el polígono guia

No se va a entrar en detalle en la explicación del proceso de matemático que implica el cálculo de Splines cúbicas, baste decir que para calcular la curva entre los nodos i e i+1 de la trayectoria, necesitaremos los nodos inmediatamente anterior y posterior (nodos i-1 e

i+2 respectivamente) para poder construir el polinomio-guía (*Figura 6.16*).

Una vez establecidos los puntos necesarios, tenemos la siguiente función interpoladora:

$$x_i(t) = C_1(t) \cdot v_{i-1} + C_2(t) \cdot v_i + C_3(t) \cdot v_{i+1} + C_4(t) \cdot v_{i+2}$$

Donde $\{v_i\}_{1 \le i \le m-1}$ para m el número total de nodos de la trayectoria, son los vectores de los nodos de la trayectoria, y C(t) para $t \in [0,1]$ los coeficientes en cada tramo que se extraen de aplicar las condiciones de contorno a los polinomios cúbicos.

$$C_1(t) = \frac{-t^3 + 3 \cdot t^2 + 3 \cdot t + 1}{6}$$

$$C_2(t) = \frac{3 \cdot t^3 - 6 \cdot t^2 + 4}{6}$$

$$C_3(t) = \frac{-3 \cdot t^3 + 3 \cdot t^2 + 3 \cdot t + 1}{6}$$

$$C_4(t) = \frac{t^3}{6}$$

Si se expresa en notación matricial, se obtiene la expresión 6.1:

$$x_i = \left[t^3, t^2, t, 1\right] \cdot \frac{1}{6} \cdot \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 0 & 3 & 0 \\ 1 & 4 & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} v_{i-1} \\ v_i \\ v_{i+1} \\ v_{i+2} \end{bmatrix} \qquad (6.1)$$

Que es la formula final que se utiliza para el cálculo de la Spline.

Como se puede observar en la *Figura6.16*, la curva resultante no pasa por los puntos de la trayectoria sino que se mantiene dentro del polinomio interpolador formado por ellos. Pero en este caso concreto SI interesa que la curva generada pase por el punto inicial y final de esta. Para cumplir esto debemos de crear 2 nodos ficticios (uno inicial y otro final).

Si en la expresión *(6.1)* con la que se calcula la Spline, se le da a t el valor de cero, queda la siguiente expresión:

$$x_i = \frac{1}{6} \cdot \left(v_{i-1} + 4 \cdot v_i + v_{i+1} \right) \qquad (6.2)$$

Ahora lo que se pretende es que x_1 (primer punto de la Spline) se corresponda con v_1 (primer punto de la trayectoria), por tanto:

$$v_1 = \frac{1}{6} \cdot \left(v_0 + 4 \cdot v_1 + v_2 \right) \qquad (6.3)$$

Se despeja el vector v_0 que es el nodo anterior al primer nodo de la trayectoria:

$$v_0 = 2 \cdot v_1 - v_2 \qquad (6.4)$$

De un modo muy parecido, pero con t=1 y asimilando x_i con v_m, siendo este el último punto de la trayectoria se obtiene:

$$v_{m+1} = 2 \cdot v_m - v_{m-1} \qquad (6.5)$$

Ahora que ya se ha calculado cual serían los nodos ficticios, deben de añadirse a los nodos de la trayectoria que se pretende alisar. El vector de la expresión *(6.4)* como primer nodo de la trayectoria, y el de la expresión *(6.5)* como el último.

Si se aplica este método de alisamiento a la trayectoria que se ha obtenido después de todos los pasos anteriores, la curva que tiene que seguir el robot es la de la *Figura 6.17*.

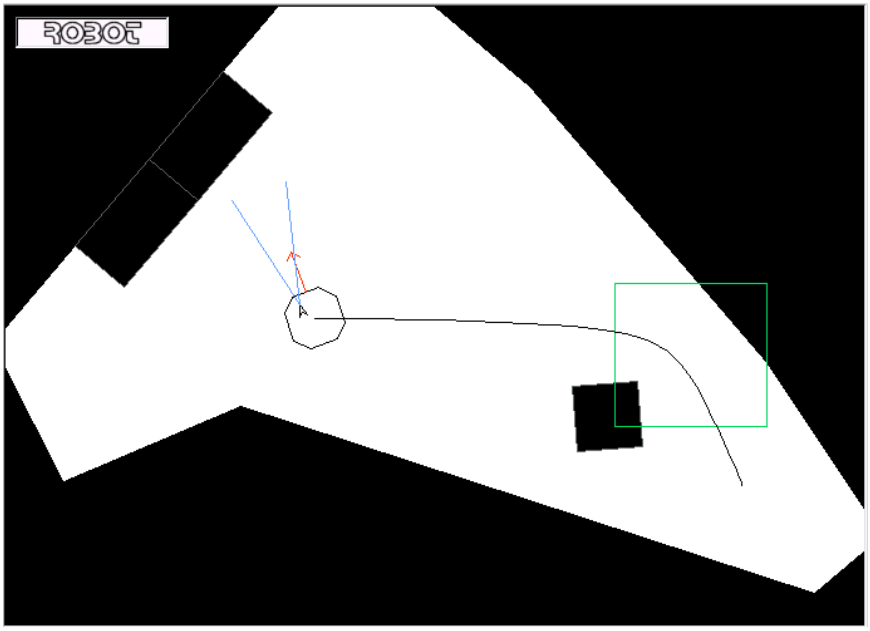


Figura 6.17: Imagen de la curva obtenida mediante la técnica de interpolación por Splines. Como se puede observar, la curva SI que pasa por el punto de origen y el de destino.

6.2.3. SEGUIMIENTO DE LA TRAYECTORIA ESCOGIDA

El fin último de generar una trayectoria es conseguir que el robot se desplace de un punto al otro del plano evitando colisionar con los objetos.

Para conseguir que esto sea posible se ha descompuesto la curva en una sucesión de puntos, más o menos separados, llamados puntos de control. La intención es utilizar el comportamiento `sfAttendAtPos` (ver Capítulo 5, apartado 5.3.1. Control por comportamiento para más detalles), para ordenarle al robot que se desplace uno por uno, a lo largo de todos los puntos de control que se han definido, mientras que al mismo tiempo se consulta constantemente la posición y orientación del robot para poder actualizar las de su homónimo virtual en el mapa. Esto implica que se tienen que realizar dos tareas en paralelo, el desplazamiento del robot de un punto de control al siguiente, y el bucle encargado de consultar la posición del robot. Para realizar ambas operaciones en paralelo, han tenido que implementarse Threads, que es una técnica de programación que permite hacer funcionar partes de código en paralelo al funcionamiento normal del programa, siempre y cuando se pueda controlar el flujo de este

A fin de que el usuario pueda comparar fácilmente la trayectoria realmente seguida por el robot con la curva generada, este va dejando a su paso un rastro de punto rojos (*Figura 6.18*).

La interfaz, una vez que el robot a alcanzado el punto de destino, pregunta al usuario si quiere almacenar en disco la imagen que vemos en el mapa, antes de este borre la trayectoria seguida.

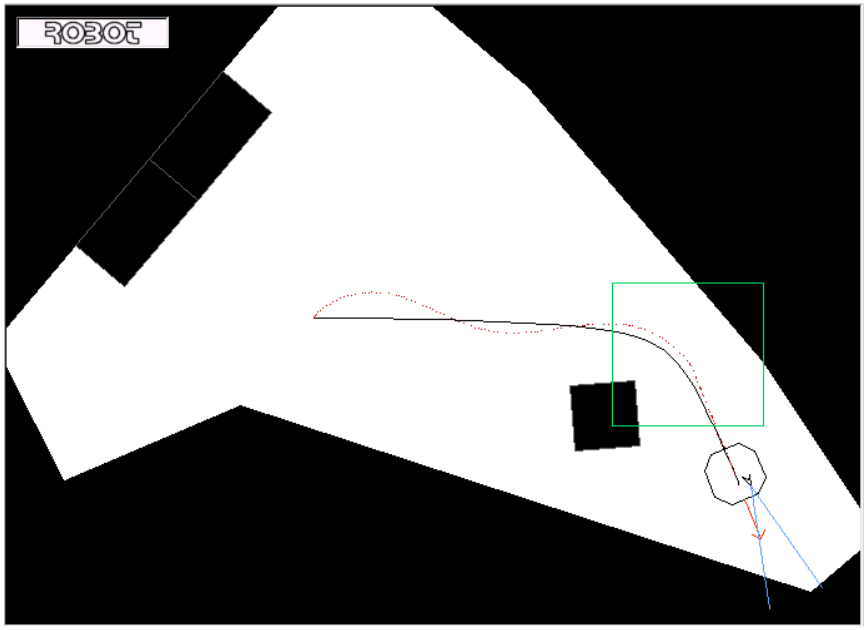


Figura 6.18: Seguimiento de la curva trazada por parte del robot, a una velocidad de 300 mm/seg, y sin comportamientos activados.

Como se puede observar, sobretodo al inicio del movimiento, el robot no sigue la curva exactamente. Esto se debe a diversos factores, uno de los cuales es que el robot comienza a "andar" antes de encararse hacia los puntos de control. En un principio, se podría haber orientado el robot en la dirección del movimiento antes de comenzar a seguir la curva, pero de lo que se trataba era de conseguir un movimiento lo más real posible, por ese motivo deliberadamente se dejo que el robot siguiera la curva como pudiese sin orientarse primero. Otra de las causas que influyen en la desviación del robot de su trayectoria, es la velocidad con que esta es seguida. Debe tenerse en cuenta que al no estar perfectamente encarado en la dirección del movimiento, cuanta más velocidad tenga el robot, más se aleja de la curva a seguir durante el transitorio de inicio

7 RESULTADOS

7.1. INTRODUCCIÓN

En este capítulo se exponen los resultados que ofrece la interfaz en todos los aspectos. Se ofrecerán diversos ejemplos de como la velocidad del robot, y los comportamientos, afectan al seguimiento de una curva, así como imágenes capturadas con la cámara de navegación, variación de parámetros de los comportamientos en tiempo real, fiabilidad del algoritmo de generación de trayectorias, y en general pruebas que demuestren el cumplimiento de los objetivos marcados en el proyecto.

7.2. PRUEBAS DE RENDIMIENTO DE LA APLICACIÓN

Las pruebas de funcionamiento del Navegador se realizaron en un ordenador Pentium III a 600 MHz con 128 Mb de RAM, con sistema operativo Windows NT Server.

En primer lugar se comprobará el tiempo que tarda la aplicación en repintar la escena en función del número máximo de lecturas de sonar a representar y del tipo de representación del mapa (centrado en el mundo, o centrado en el robot). La prueba se realiza en una habitación cerrada, calculando el tiempo que tarda el PC en actualizar la vista del mapa haciendo la media de 300 pruebas. En un principio no se tendrá en cuenta la posibilidad de cargar un mapa al navegador porque lo que se pretende es comprobar como afecta la limitación del número de lecturas de sonar al tiempo de representación. Los resultados numéricos obtenidos (medidos en segundos) están representados en la tabla de la página siguiente:

	Mapa centrado en el Mundo	Mapa centrado en el Robot

30 lecturas	0.035	0.074
40 lecturas	0.043	0.075
50 lecturas	0.045	0.078
60 lecturas	0.0482	0.0841
70 lecturas	0.0548	0.0845
80 lecturas	0.0551	0.0862
90 lecturas	0.0622	0.0903
100 lecturas	0.064	0.0923

Si se hace la inversa de estos resultados (número de actualizaciones del mapa por segundo) y se representan gráficamente los resultados, obtenemos la gráfica de la *Figura 7.1.*:

Es evidente cual de los dos métodos de representación es más lento, pero estos

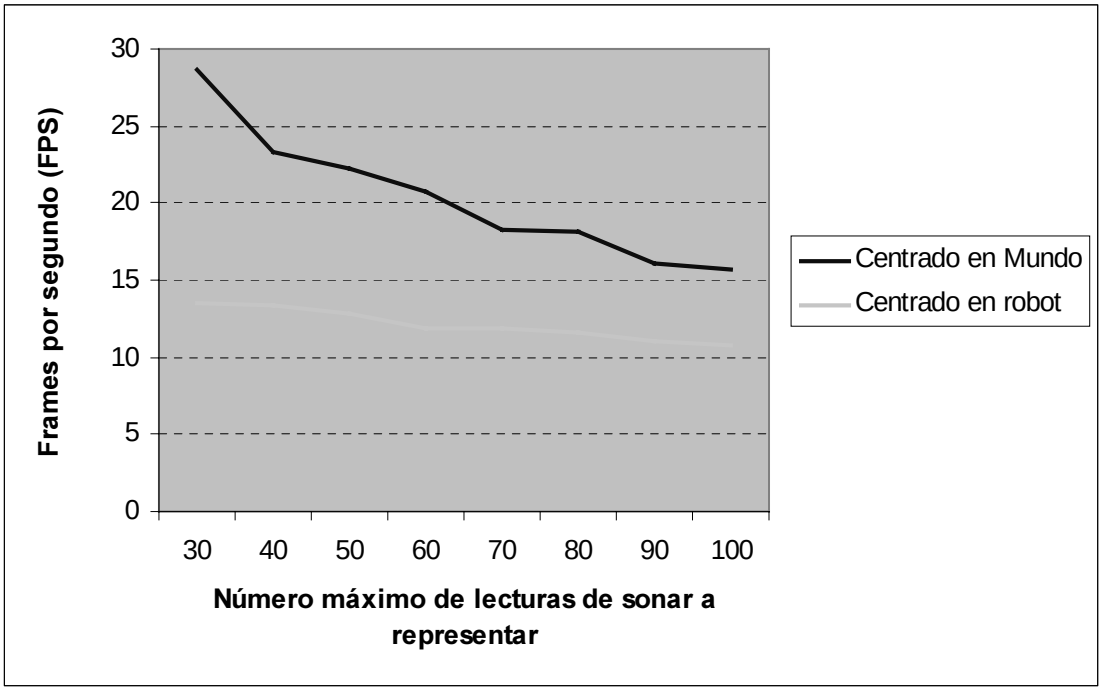


Figura 7.1: Número de frames por segundo (actualizaciones del mapa por segundo) obtenidos, en función del número máximo de lecturas de sonar y del tipo de representación del mapa.

resultados eran previsibles. Cuando la representación del mapa esta centrada en el mundo, solo se calculan los giros y traslaciones del robot, mientras que en el caso de la representación centrada en el robot, deben calcularse los giros y traslaciones del robot y de las lecturas de los sonares. Pero esta diferencia de rendimiento se hace todavía más evidente si se carga un mapa predefinido en el navegador y se realiza el mismo tipo de pruebas:

	Mapa centrado en el Mundo	Mapa centrado en el Robot
30 lecturas	0.035	0.123
40 lecturas	0.036	0.128
50 lecturas	0.0372	0.131
60 lecturas	0.0402	0.137
70 lecturas	0.041	0.14
80 lecturas	0.044	0.1405
90 lecturas	0.0465	0.146
100 lecturas	0.0525	0.147

De nuevo se calculan las inversas y se grafican los resultados, obteniendo:

En esta ocasión la diferencia de rendimiento entre los dos métodos de representación es mucho más evidente. Mientras que cuando la representación está

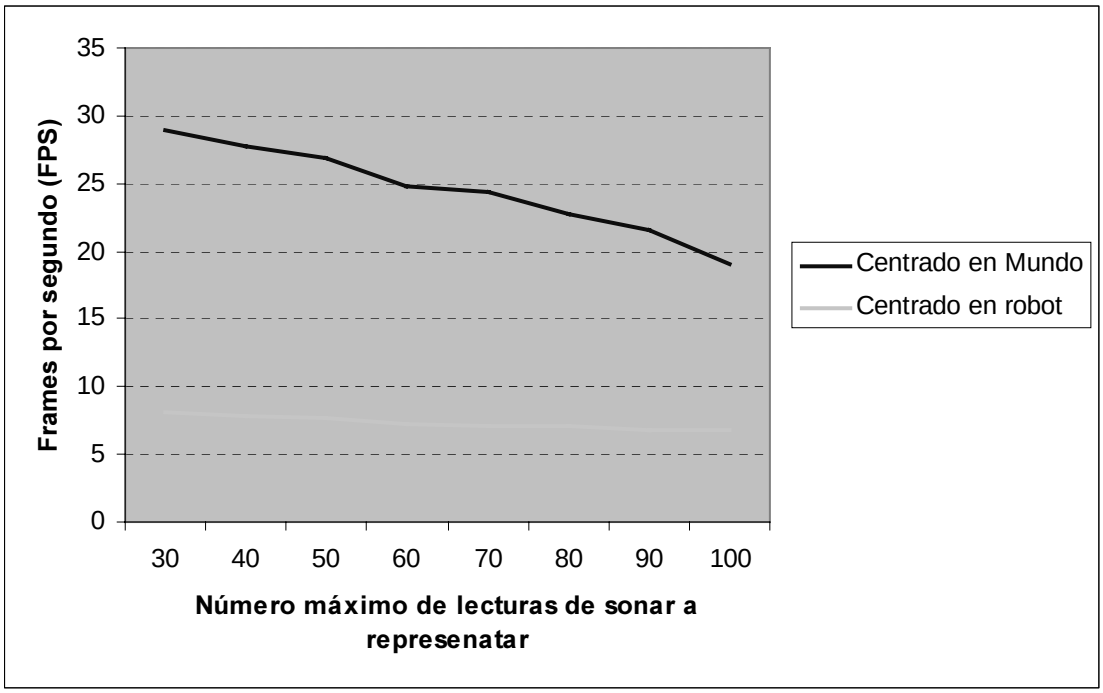


Figura 7.2: Número de frames por segundo obtenidos en función del número máximo de lecturas de sonar y del tipo de representación del mapa. Esta vez con un mapa predefinido cargado en la aplicación

centrada en el mundo, este no tiene que redibujarse (capítulo 4, apartado 4.3.4.), en el caso de la representación centrada en el robot, el mundo tiene que ser recalculado constantemente con la consiguiente carga de trabajo que ello representa

De cualquier forma, en el momento de realizarse estas pruebas, se produjo un resultado inesperado al comparar los datos de los dos modos de representación sin mapa, con los datos de las mismas representaciones pero con mapa (ver *Figura 7.3* y *7.4*):

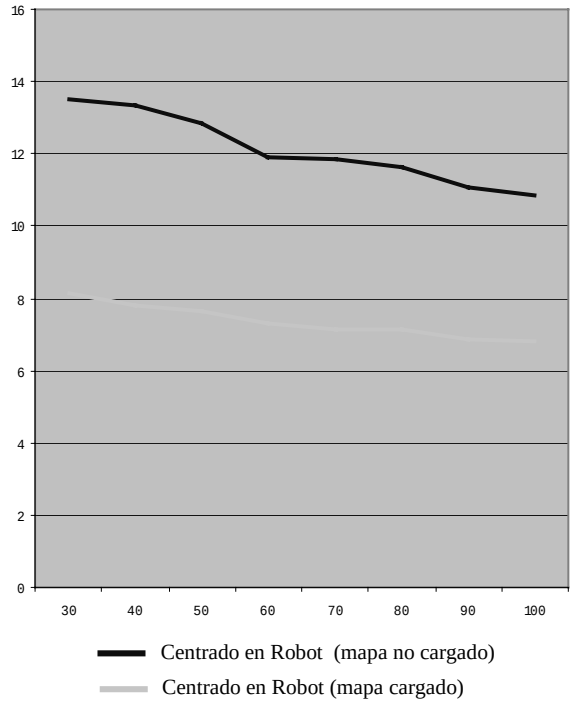


Figura 7.3

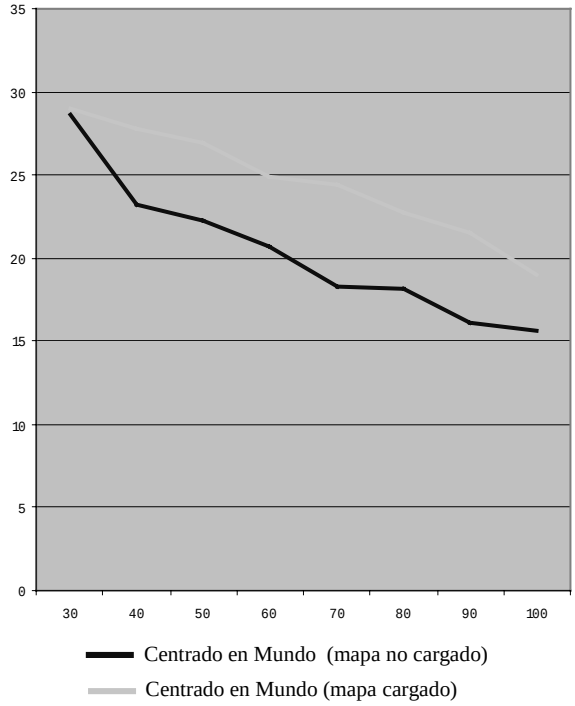


Figura 7.4

La *Figura 7.3* muestra los resultados de la represtación centrada en el robot, dos gráficas muy similares desplazadas una respecto a la otra debido a la diferencia de rendimiento que provoca tener que realizar los cálculos adicionales del mapa cargado. Lo realmente sorprendente es la gráfica de la *Figura 7.4*. Cuando la representación del mapa se realiza centrada en el mundo, el mundo en sí esta representado en un "buffer" interno y simplemente se tiene que copiar al "buffer" de representación (ver de nuevo el apartado 4.3.4), tenga o no un mapa cargado y representado. Normalmente el rendimiento habría de ser prácticamente idéntico en los dos casos, pero la sorpresa es que no solo el rendimiento es diferente, sino que es superior (se obtienen más frames por segundo) en caso de tener un mapa cargado. La única explicación que puede justificar este hecho es que las librerías gráficas MIL, cuando copian la información de un "buffer" a otro, copian solo los píxeles que poseen un valor diferente de cero (negro o ausencia de color), por lo que un "buffer" que contenga un mapa predefinido siempre se copiará más rápido que un "buffer" en blanco, en cuyo caso deben copiarse todos los píxeles.

En conclusión: en todos los casos es mejor elegir como método de representación el centrado en el mundo.

7.3. PRUEBAS DEL FUNCIONAMIENTO DE LOS COMPORTAMIENTOS

En este capítulo se ha intentado poner a prueba la aplicación en el manejo de los comportamientos predefinidos por Saphira, además de comprobar cuales son más útiles, en que situaciones y cuales sus valores más adecuados.

Como ya se comentó en su momento (ver apartado *5.3.2 Control por comportamientos*) el Navegador permite activar y desactivar comportamientos predefinidos por Saphira, además de modificar sus parámetros sin necesidad de desactivarlos. Para determinar cuales son los valores más adecuados para estos parámetros, se ha preparado el siguiente experimento: situando el robot en un pasillo, se ha procedido ha bloquear el paso y se ha dejado una puerta abierta, de manera que la única opción lógica del robot sería dar media vuelta o entrar por la puerta. A fin de que pueda apreciarse en las imágenes cual es la trayectoria seguida por el robot, este irá dejando a su paso una serie de puntos rojos que, por un lado permitirá apreciar fácilmente el camino seguido, y por otro estimar la velocidad del robot en función de la separación que tengan los puntos entre sí.

Para esta primera prueba se fijaron los parámetros de evitar colisión y mantener distancias a sus máximos correspondientes, elevando la seguridad del robot.

<table>
<tr><th></th><th>Evitar Colisión</th><th>Mantener Distancias</th><th>Velocidad constante</th></tr>
<tr><td>Prioridad</td><td>0</td><td>1</td><td>2</td></tr>
<tr><td>TimeOut</td><td>0</td><td>0</td><td>0</td></tr>
<tr><td>Sensibilidad frontal</td><td>3</td><td rowspan="4"></td><td rowspan="4"></td></tr>
<tr><td>Sensibilidad lateral</td><td>3</td></tr>
<tr><td>Ganancia giro</td><td>5</td></tr>
<tr><td>Radio seguridad</td><td>100</td></tr>
<tr><td>Velocidad de precaución</td><td rowspan="3"></td><td>30</td><td rowspan="3"></td></tr>
<tr><td>Sensibilidad a los obstáculos</td><td>2</td></tr>
<tr><td>Velocidad constante</td><td></td><td>100</td></tr>
</table>

La secuencia obtenida fue:

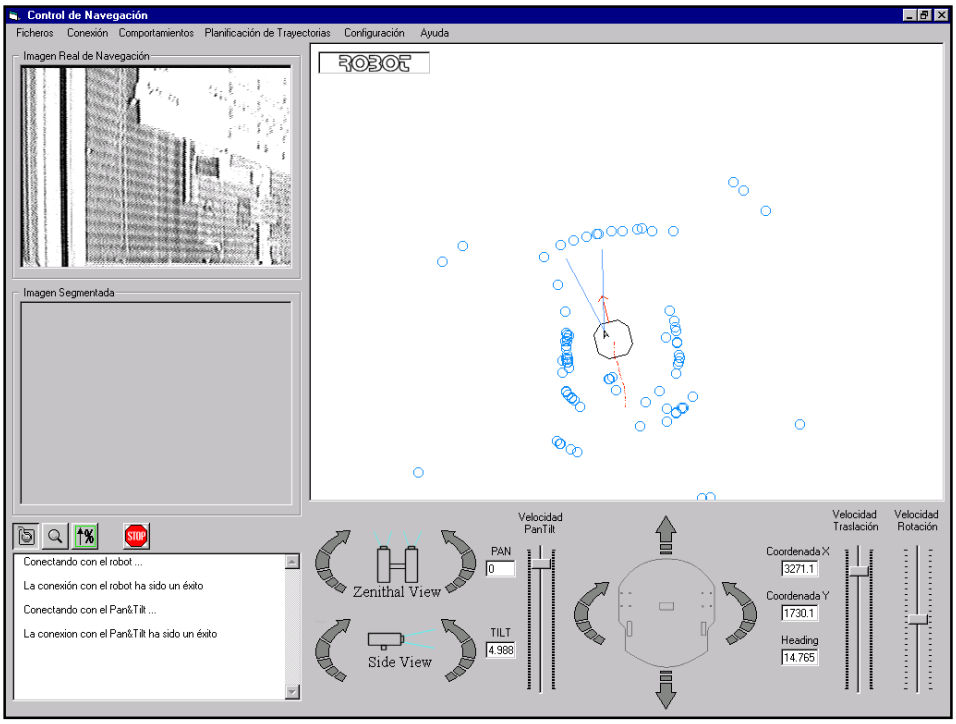


Figura 7.5a

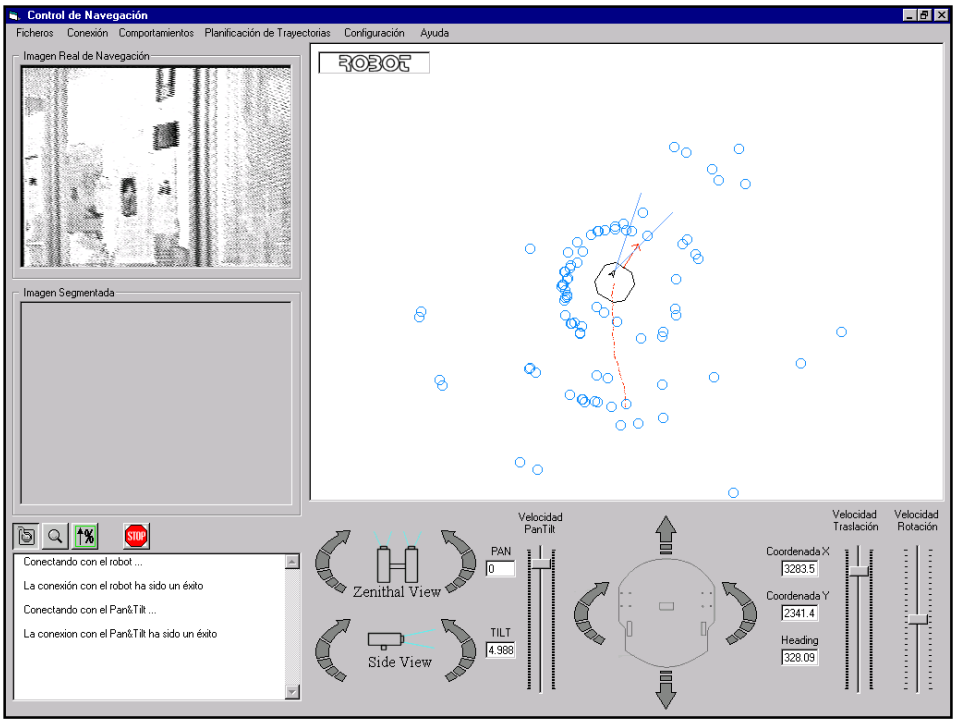


Figura 7.5b

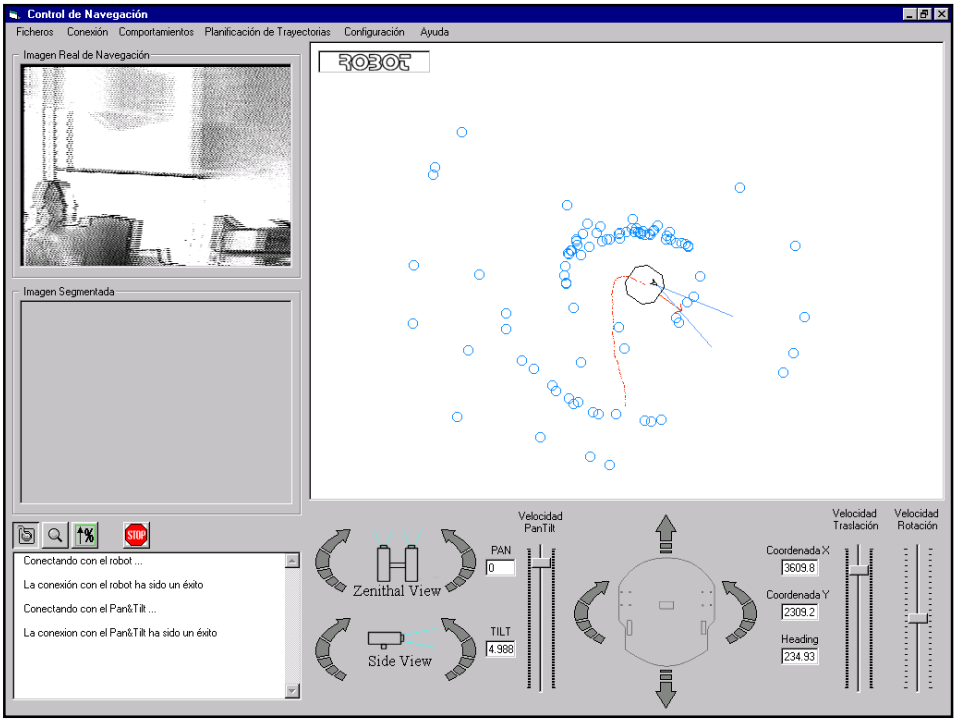


Figura 7.5c

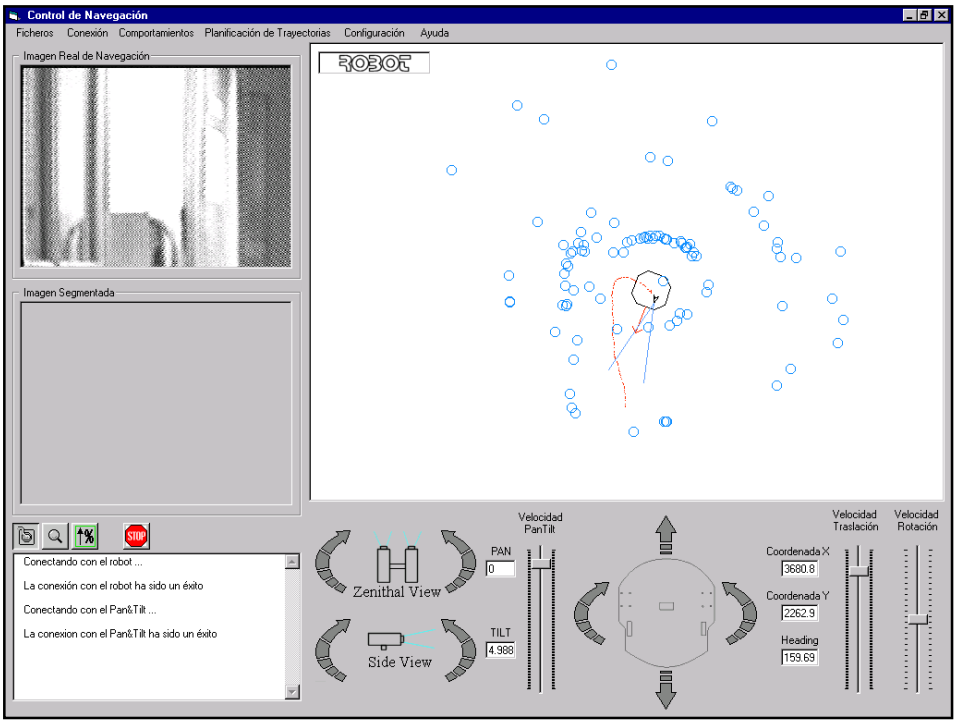


Figura 7.5d

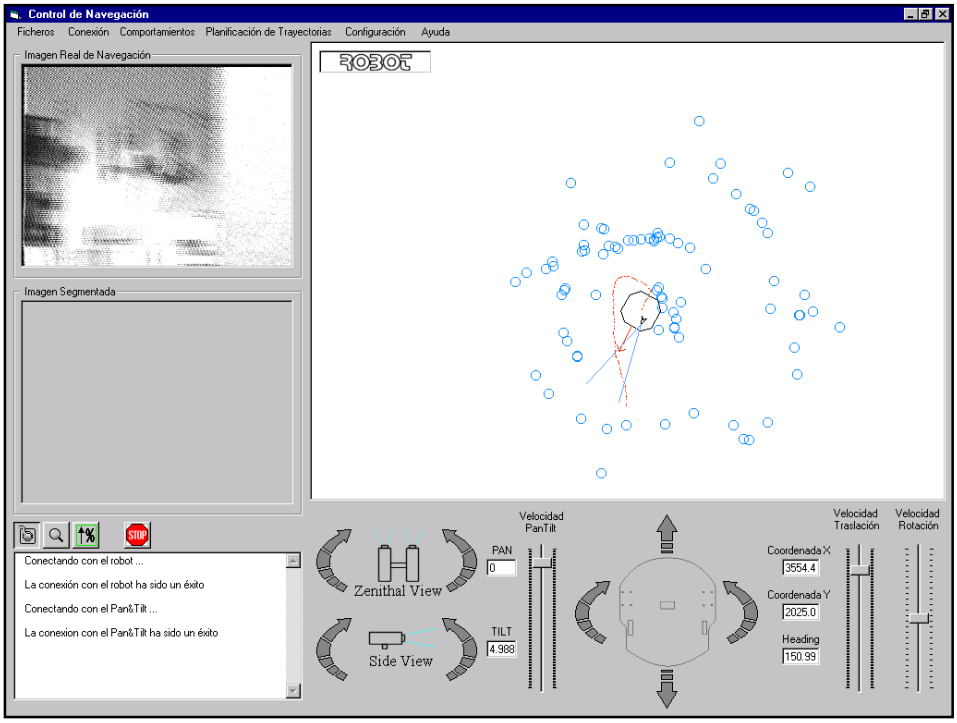


Figura 7.5e

Como se puede apreciar en la secuencia de imágenes, el robot no llega a colisionar con el obstáculo del pasillo (*Figura 7.5c*), pero los primeros problemas aparecen cuando se encara la puerta. Debido a los parámetros tan conservadores que han sido fijados en los comportamientos de evitar colisión y mantener distancias, el robot considera que el espacio que tiene el marco de la puerta no es suficiente para pasar, y por tanto no lo hace (*Figura 7.5d*).

A pesar de que la *Figura 7.5e* muestra lecturas de sonar muy cerca del robot, este no llega a colisionar con el marco de la puerta, ya que llega a detenerse completamente antes de que suceda. El cúmulo de lecturas de sonar situado inmediatamente detrás del robot, son lecturas imprecisas de los sonares traseros, al igual que sucede en las *Figuras 7.5a y 7.5b* donde pueden apreciarse lecturas en mitad del pasillo.

Aunque es vital que el robot goce de cierta seguridad cuando se desplaza, lo que no es aceptable es que sea incapaz de entrar por una puerta, por lo que en vista de los resultados obtenidos, se decide relajar un poco los parámetros de seguridad del robot. En la prueba siguiente los parámetros establecidos para los comportamientos son:

	Evitar Colisión	Mantener Distancias	Velocidad constante
Prioridad	0	1	2
TimeOut	0	0	0
Sensibilidad frontal	2		
Sensibilidad lateral	2		
Ganancia giro	5		
Radio seguridad	100		
Velocidad de precaución		30	
Sensibilidad a los obstáculos		2	
Velocidad constante			100

La secuencia obtenida fue:

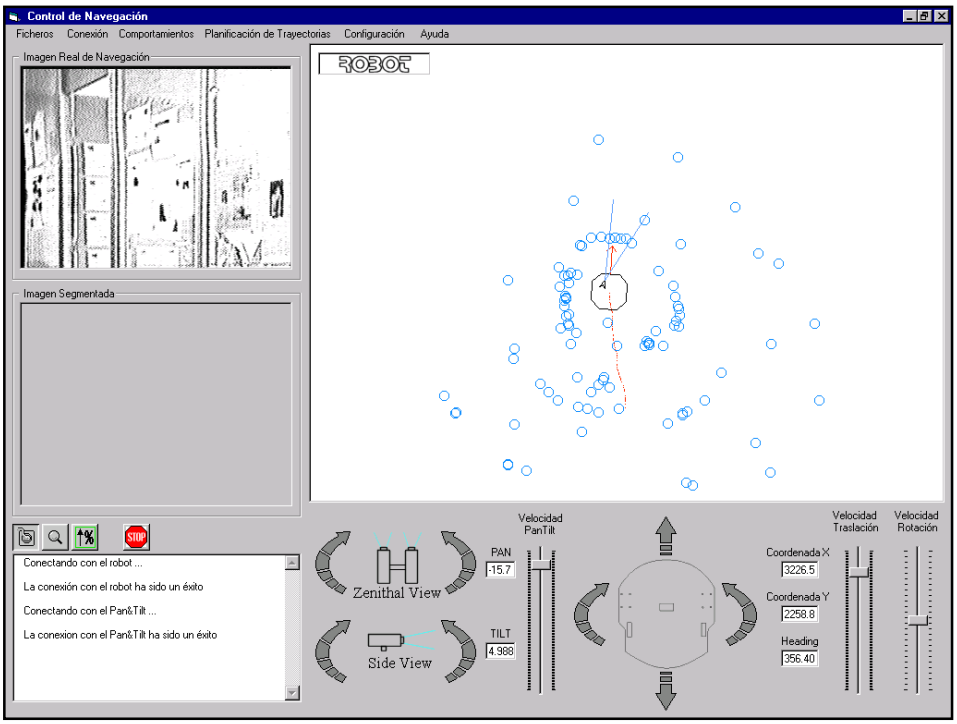


Figura 7.6a

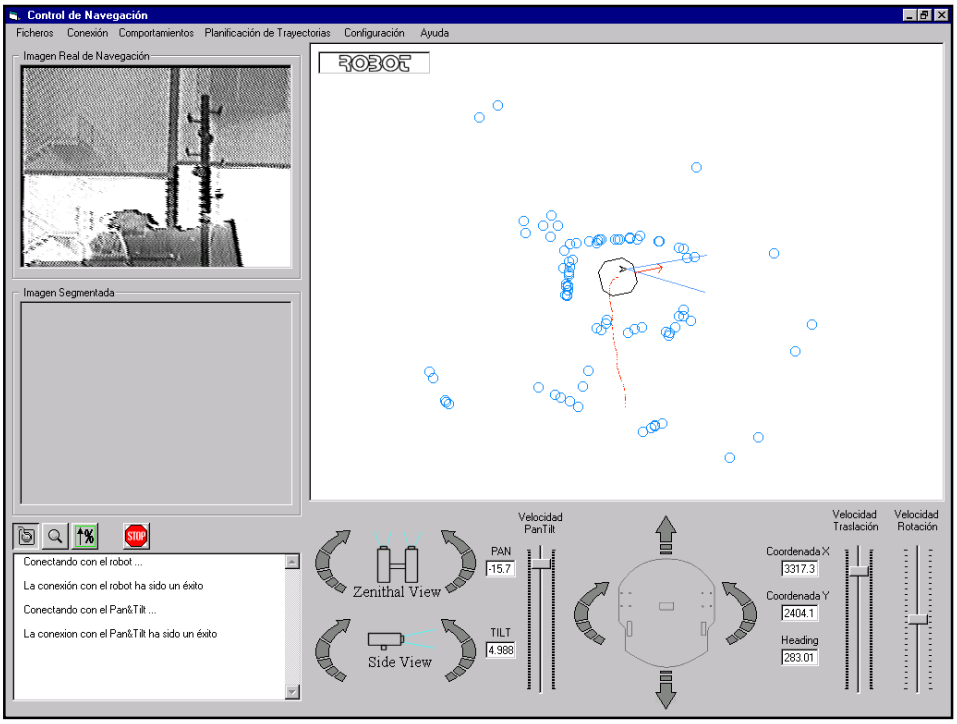


Figura 7.6b

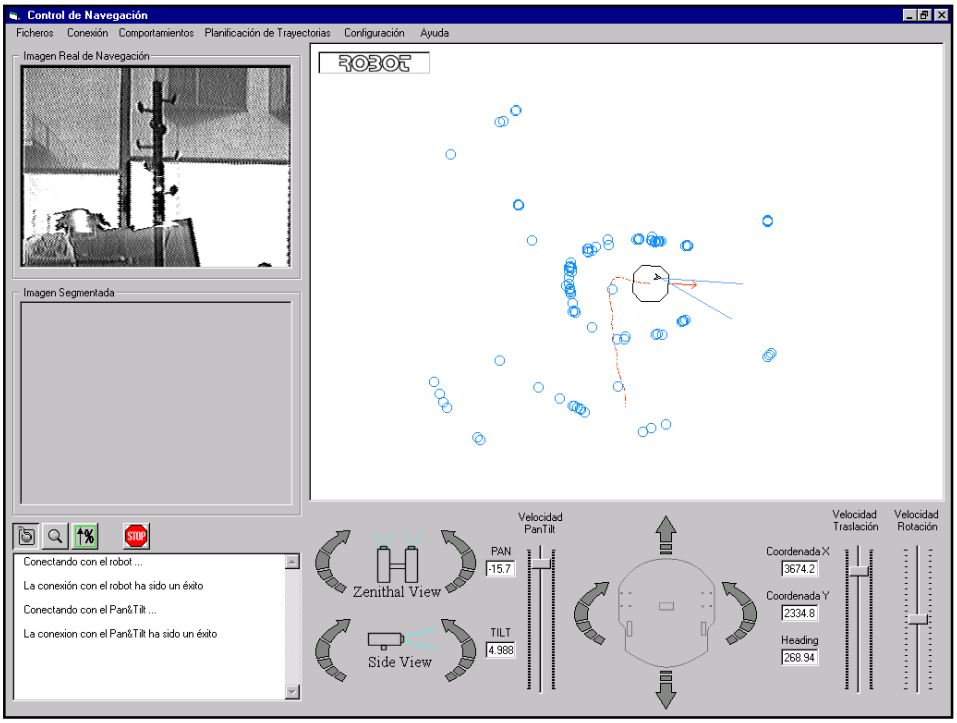


Figura 7.6c

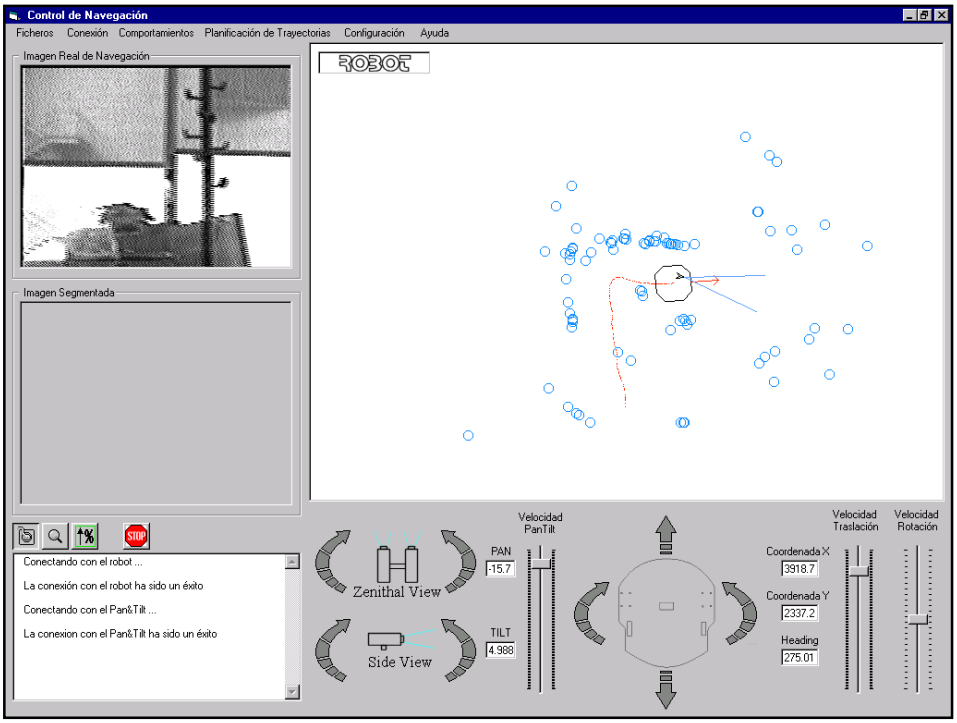


Figura 7.6d

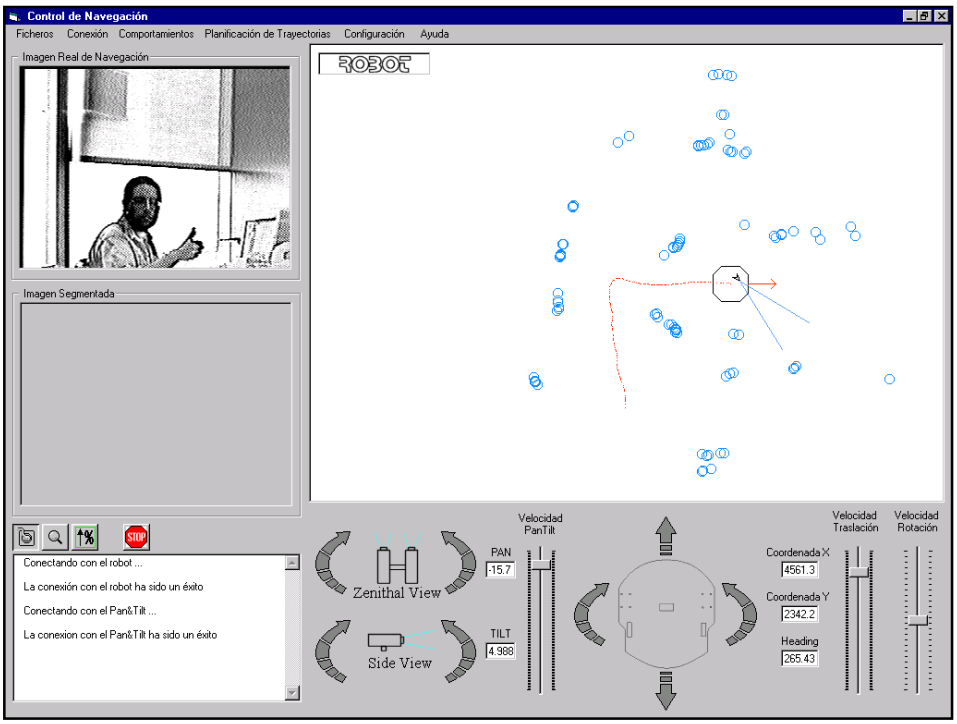


Figura 7.6e

En esta ocasión la respuesta del robot es perfecta. Avanza por el pasillo sin dificultad (*Figura 7.6a*) hasta que llega a una distancia prudencial del obstáculo (*Figura 7.6b*), momento en que comienza a girar y encara la puerta. En las *Figuras 7.6c* y *7.6d* se aprecia como los sonares detectan perfectamente los marcos de la puerta, propiciando que el robot se reoriente ligeramente para entrar por el centro de ella. Finalmente el robot entra sin ningún tipo de problemas en la habitación, dejando atrás la puerta (ver *Figura 7.6e*).

También en esta ocasión se pueden advertir lecturas falsas de los sonares detrás del robot, a lo largo de toda la secuencia. Las reiteradas lecturas falsas detrás del robot , siempre muy cerca de él, lleva a la conclusión que probablemente alguno de los sonares del anillo trasero está mal calibrado y se refleja contra el suelo, provocando estas lecturas falsas inmediatamente detrás del robot.

El éxito del experimento anterior revela que quizás se habían configurado los parámetros de los comportamientos con excesivo empeño en la seguridad del robot, lo cual provocó un resultado poco satisfactorio en el primer experimento. Por ese motivo, en este tercer experimento se ha optado por relajar aún más los parámetros de seguridad de los comportamientos. Desactivando completamente el comportamiento *Mantener Distancias*, se pretende averiguar hasta que punto mejora o empeora la navegación segura del robot. En esta ocasión, los parámetros fijados para los comportamientos son:

Obteniendo la siguiente secuencia de imágenes:

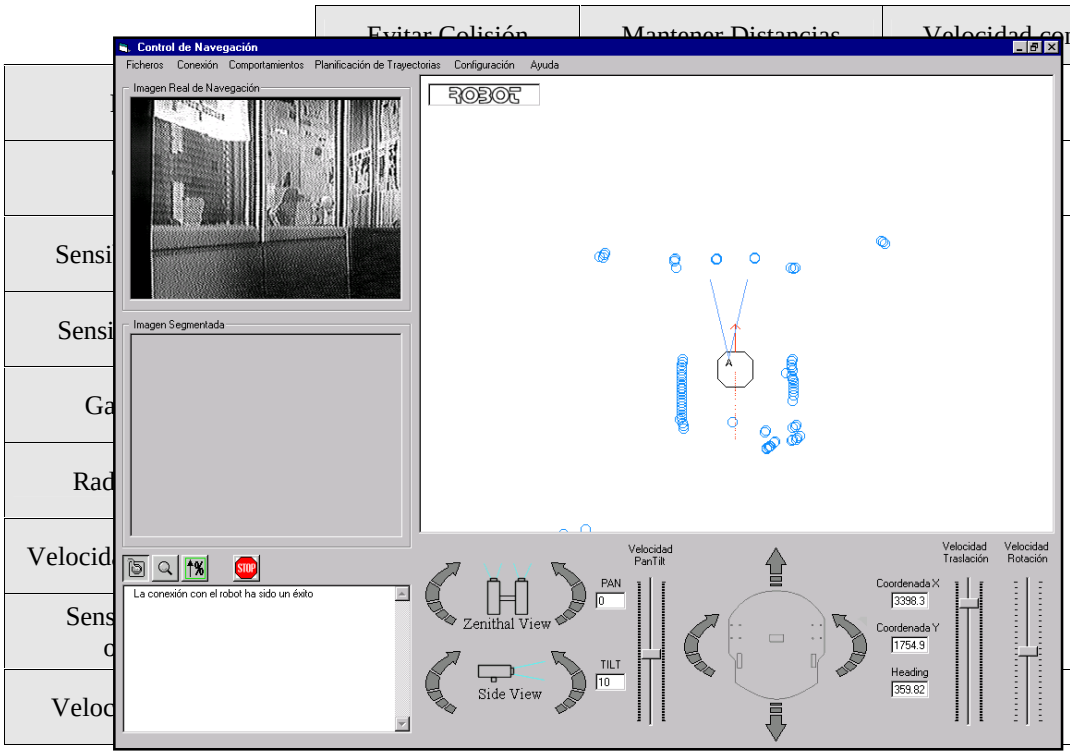


Figura 7.7a

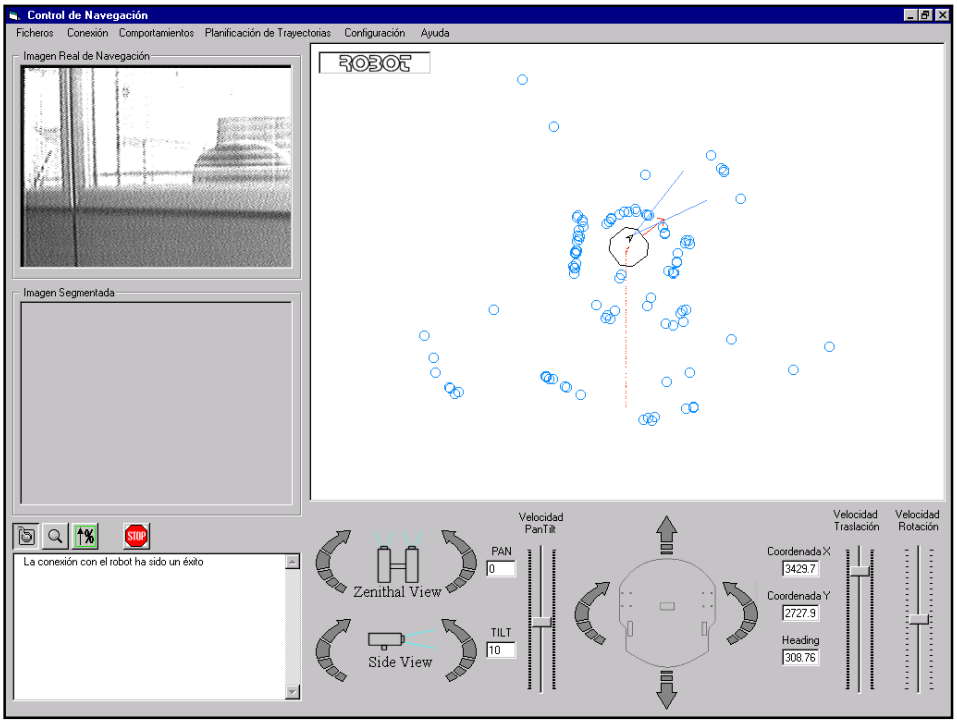
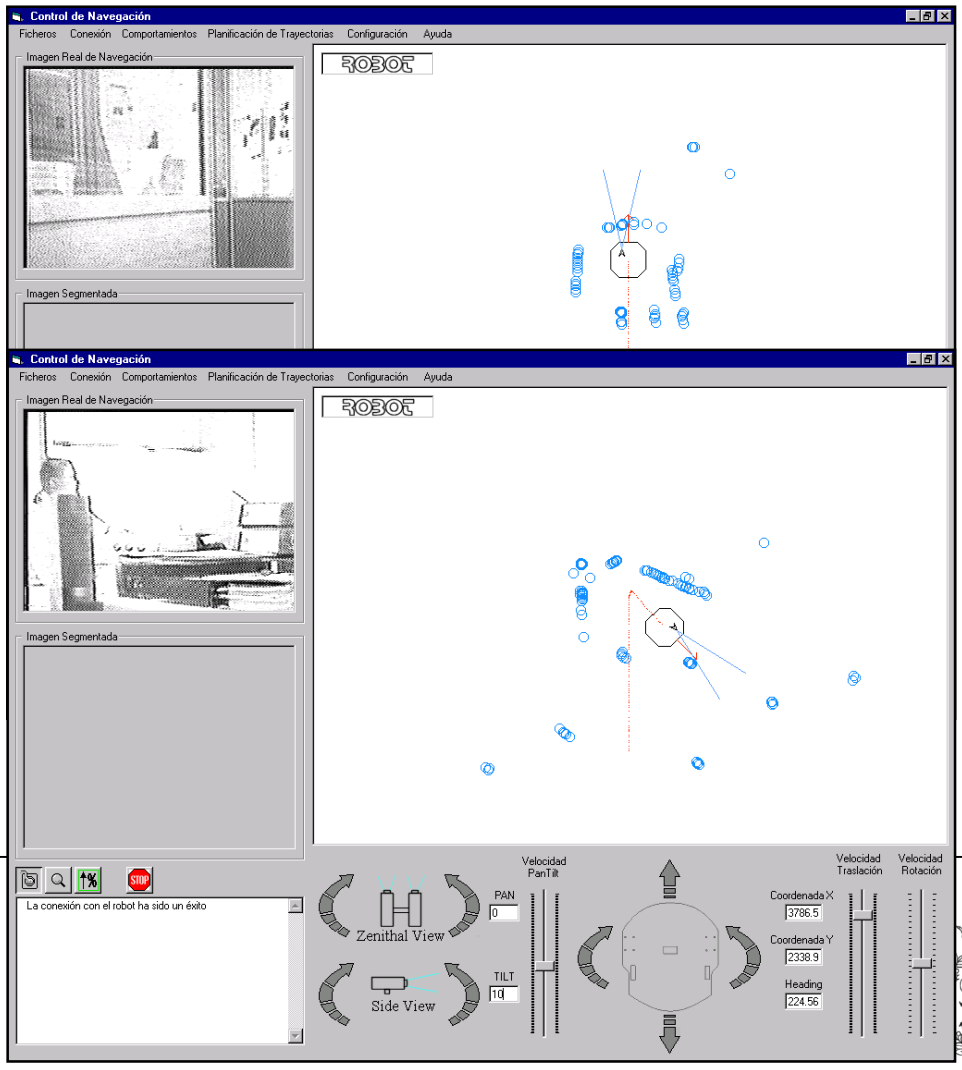

Control de Navegación
Ficheros Conexión Comportamientos Planificación de Trayectorias Configuración Ayuda
Imagen Real de Navegación
Imagen Segmentada
ROBOT
La conexión con el robot ha sido un éxito
Zenithal View
Side View
PAN
TILT
Velocidad PanTilt
Velocidad Traslación
Velocidad Rotación
Coordenada X
Coordenada Y
Heading

Figura 7.7c


Control de Navegación
Ficheros Conexión Comportamientos Planificación de Trayectorias Configuración Ayuda
Imagen Real de Navegación
Imagen Segmentada
ROBOT
La conexión con el robot ha sido un éxito
Zenithal View
Side View
PAN
TILT
Velocidad PanTilt
Velocidad Traslación
Velocidad Rotación
Coordenada X
Coordenada Y
Heading

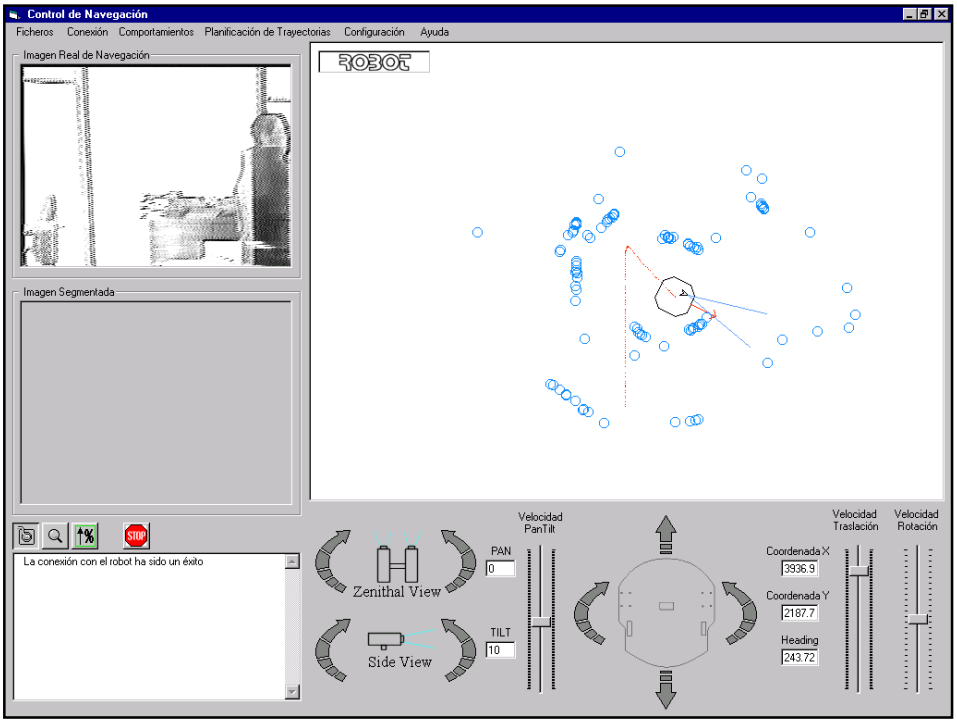

Control de Navegación
Ficheros
Conexión
Comportamientos
Planificación de Trayectorias
Configuración
Ayuda
Imagen Real de Navegación
Imagen Segmentada
ROBOT
La conexión con el robot ha sido un éxito
Zenithal View
Side View
PAN
0
TILT
10
Velocidad PanTilt
Velocidad Traslación
Velocidad Rotación
Coordenada X
3936.9
Coordenada Y
2187.7
Heading
243.72

Figura 7.7e

Aunque quizás no se puede apreciar bien en la secuencia de imágenes, el resultado es el peor de todos los obtenidos hasta el momento. Lo primero que llama la atención es como avanza el robot. En los experimentos anteriores, la trayectoria seguida por el robot en los primeros instantes no es totalmente rectilínea, debido a las lecturas de sonar laterales. Sin embargo, en esta ocasión el robot avanza totalmente recto sin detenerse ni desviarse lo más mínimo de su trayectoria (ver *Figuras 7.7a* y *7.7b*).

A pesar que el obstáculo del pasillo es detectado por los sonares desde el primer momento (ver *Figura 7.7a*), la desconexión del comportamiento de *Mantener Distancias* provoca que el robot no disminuya su velocidad constante de avance a 100 mm/seg (ver *Figura 7.7b*), hasta que es demasiado tarde y colisiona (*Figura7.7c*). Afortunadamente el obstáculo del pasillo consistía en una pared de cartón puesto que, como se puede ver claramente por las lecturas de los sonares en la *Figura 7.7c*, no solo colisiona con él, sino que llega a empujarlo y deformarlo antes de conseguir modificar su orientación lo suficiente como para alejarse. Una vez encarada la puerta (ver *Figura 7.7d*) los resultados no son mejores. De nuevo el robot intenta reorientarse demasiado tarde, por lo que cuando intenta esquivar el marco de la puerta, ya a colisionado con él y le es imposible alejarse, quedando encallado (*Figura 7.7e*).

Gracias a estos resultados, se hace bastante evidente la conveniencia de mantener activado el comportamiento de *Mantener Distancias* durante la navegación del robot. Un parámetro de velocidad cauta suficientemente bajo (30 mm/seg, incluso menos) permitirá al robot un acercamiento mucho más seguro a los obstáculos, sin que una elevada velocidad de traslación pueda provocar los accidentes ocurridos en este tercer experimento.

7.4 SEGUIMIENTO DE UNA TRAYECTORIA E INFLUENCIA DE LA VELOCIDAD

En este experimento se pretende poner a prueba la fiabilidad del robot en el seguimiento de trayectorias, así como la influencia de la velocidad en los resultados obtenidos.

Para este experimento se tuvo que fijar una trayectoria modelo con el fin de poder comparar equitativamente el resultado del seguimiento de la misma a diferentes

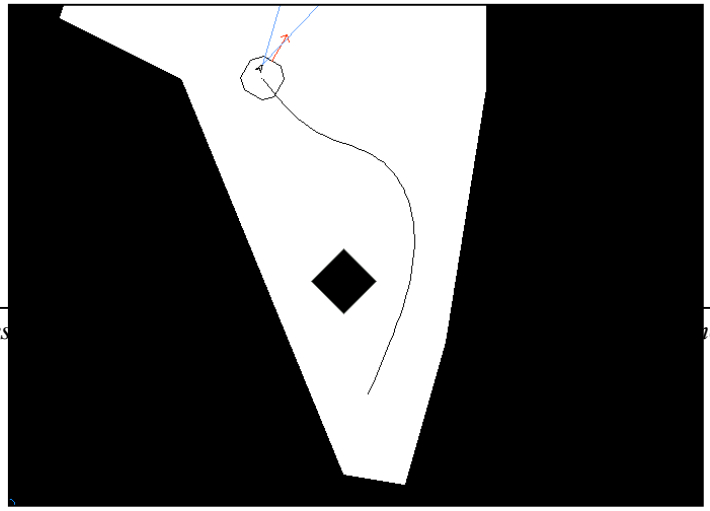


Figura 7.8: Camino a seguir por el robot en el experimento

velocidades. La trayectoria elegida es la de la *Figura 7.8*.

Para constatar el camino seguido por el robot, se ha aprovechado la utilidad que brinda el propio Navegador de almacenar en un fichero imagen el resultado del seguimiento de una trayectoria.

Nótese que aunque el Navegador da la posibilidad de modificar la velocidad de traslación del robot en tiempo real, incluso durante el seguimiento de una trayectoria definida, en esta ocasión la velocidad se ha mantenido constante debido al propósito del experimento.

7.4.1 SEGUIMIENTO DE LA TRAYECTORIA A 100 MM/SEG

En este caso se establece la velocidad de traslación a 100 milímetros por segundos.

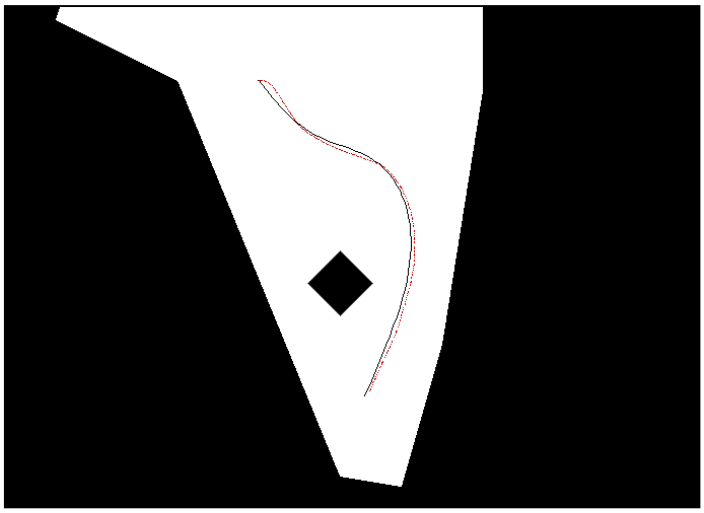


Figura 7.9: Trayectoria seguida por el robot a 100 mm/seg

Como se puede observar (*Figura 7.9*), a esta velocidad el seguimiento de la curva es muy preciso.

La zona en la que más se desvía el robot de la trayectoria es al comienzo de la misma. Esto es debido a que el robot no se orienta en la dirección de la curva antes de comenzar su seguimiento. Aunque no sería complicado orientar el robot en la dirección de la curva, se decidió no hacerlo para conseguir un movimiento del robot más "natural".

La máxima separación del robot de la trayectoria a seguir es aproximadamente 10 cm.

7.4.2 SEGUIMIENTO DE LA TRAYECTORIA A 200 MM/SEG

En esta ocasión, en el transitorio de inicio se aprecia un aumento de la desviación en la trayectoria seguida por el robot.

Este aumento en la desviación de la trayectoria se debe a la duplicación de la velocidad de seguimiento, aunque el seguimiento de la trayectoria sigue siendo satisfactorio.

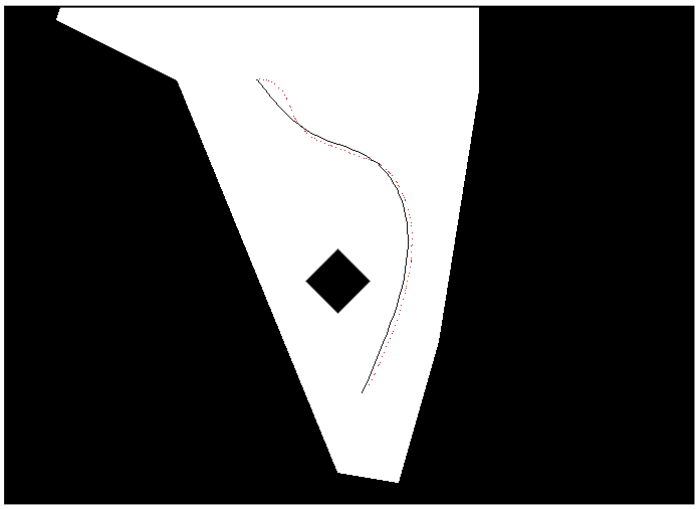


Figura 7.10: Seguimiento de la trayectoria a 200 mm/seg

7.4.3 SEGUIMIENTO DE LA TRAYECTORIA A 300 MM/SEG

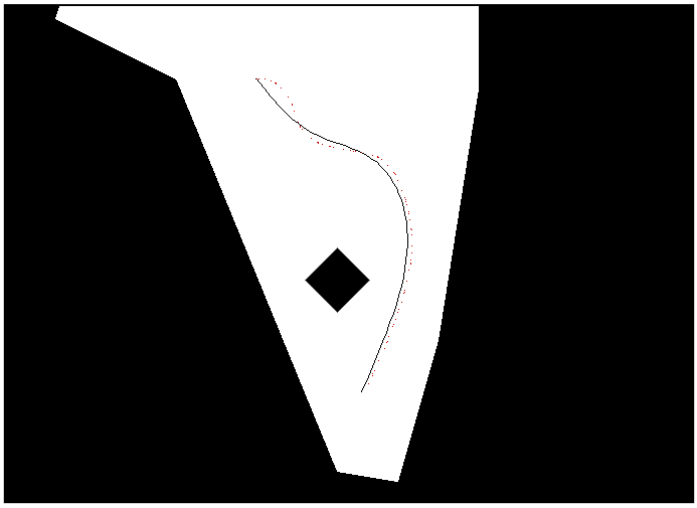


Figura 7.11: Seguimiento de la trayectoria a 300 mm/seg

Con un aumento de velocidad de 100 milímetros por segundo con respecto al experimento anterior, no se aprecia un gran cambio en el seguimiento de la trayectoria.

Lo que no se puede apreciar bien en la imagen es como el robot se frena durante el recorrido de trayectoria. Esto se debe a que la velocidad de traslación del robot es tan elevada que cuando le llega la orden de dirigirse al siguiente punto de la trayectoria, ya lo a dejado atrás y tiene tendencia a frenarse. Una solución a este problema sería aumentar la distancia entre los puntos que definen la trayectoria (Spline), pero provocaría una pérdida de definición en la misma.

7.4.4 SEGUIMIENTO DE LA TRAYECTORIA A 700 MM/SEG

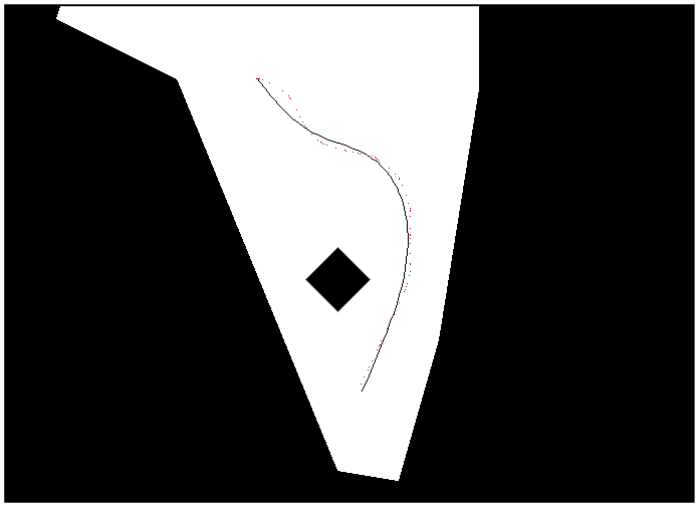


Figura 7.12: Seguimiento de la trayectoria a 700 mm/seg

Incluso a 700 milímetros por segundo el seguimiento de la trayectoria es lo suficientemente preciso como para considerarse aceptable.

Como es lógico, a esta velocidad se acentúa aún más el efecto de frenado observado en el experimento anterior, haciéndose más evidente (ver apartado 7.4.3)

7.4.5 CONCLUSIÓN

Los resultados de los experimentos demuestran que el robot es capaz de seguir eficazmente una trayectoria fijada, con relativa independencia de la velocidad de traslación. Aunque en el transitorio de inicio la diferencia entre la trayectoria a seguir y la trayectoria seguida se maximiza, en ningún caso llega a ser alarmante la diferencia.

El punto negativo de los resultados se encuentra en los "frenazos" que experimenta el robot cuando tiene que seguir una trayectoria a gran velocidad. Aún así, en trayectorias muy complejas, con curvas muy cerradas, es preferible que el robot frene para no alejarse demasiado de la trayectoria, a que la pierda.

7.5 INFLUENCIA DE LOS COMPORTAMIENTOS EN EL SEGUIMIENTO DE TRAYECTORIAS

Lo que se pretende es evaluar cual es el comportamiento del robot cuando se le obliga a seguir una trayectoria que es errónea, o imposible de seguir.

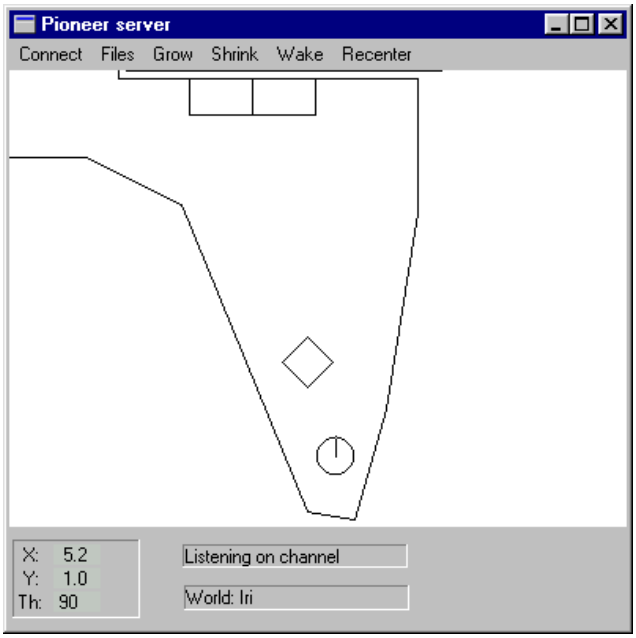


Figura 7.13: Imagen del mapa cargado en el simulador

Para conseguir este efecto, se ha cargado un mapa en blanco en el navegador y se ha realizado la conexión con el simulador del robot, que tiene cargado el mapa de la *Figura 7.13*.

Seguidamente se obliga al Navegador a generar una trayectoria rectilínea hacia adelante. Como el mapa cargado en el Navegador esta en blanco, no existe ningún tipo de restricción en el momento de generar la trayectoria.

Seguidamente, se realiza la conexión al simulador y se habilitan los comportamientos de evitar colisión y mantener distancias con sus parámetros predeterminados

Cuando el Navegador

pregunta si se desea seguir la trayectoria, se responde afirmativamente, y estos son los resultados:

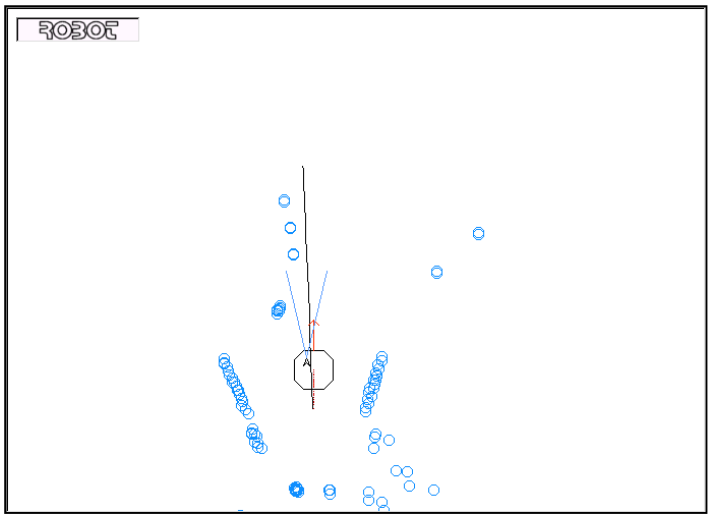
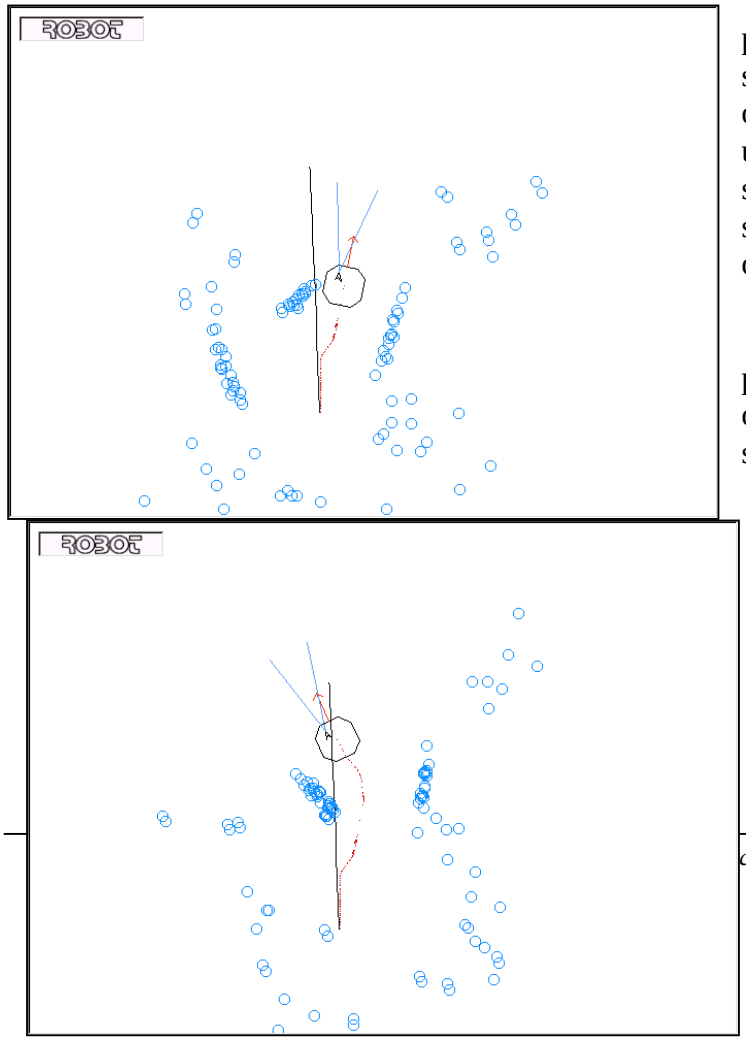


Figura 7.14.

En los primeros momentos del seguimiento de la trayectoria se aprecia como la velocidad de traslación es muy inferior a la fijada por el experimento (100 mm/seg). Este cambio en la velocidad se debe al comportamiento *Mantener Distancias*, que detecta la presencia de obstáculos cercanos y disminuye la velocidad hasta el valor que tiene fijado en su parámetro velocidad cauta (ver 5.3.1. Control por Comportamientos).



En la *Figura 7.15* se puede apreciar cual es la situación. La trayectoria definida obliga al robot a dirigirse contra un obstáculo, por lo que el robot se encuentra en el dilema de seguir la trayectoria o evitar el obstáculo.

Debido a que la prioridad de evitar la colisión con obstáculos es superior a la de seguir una trayectoria, el robot se aleja del obstáculo pero tratando de mantenerse lo más cerca posible de la trayectoria.

La *Figura 7.16* muestra como el robot a superado con éxito el obstáculo (recuérdese la imagen de la *Figura 7.13*), y se reorienta para seguir la trayectoria.

Debe quedar claro que en este experimento la situación no era crítica, es decir, aunque existía un objeto que interceptaba la trayectoria del robot, era físicamente posible evitarlo sin apartarse demasiado de la trayectoria marcada. En casos más extremos, en que el obstáculo obligase al robot a apartarse mucho de la trayectoria, o incluso que fuese físicamente imposible seguirla, el robot pone su integridad ante todo y pasado un tiempo (más o menos grande en función de la distancia que le queda al robot por recorrer antes de quedar bloqueado), abandona su intención de llegar al punto de destino.

7.6. RESULTADOS OBTENIDOS POR EL ALGORITMO DE "PATH PLANNING"

En este apartado no se tratará de mostrar resultados en cuanto a la velocidad de generación del mapa de caminos en función del método empleado, ni de la influencia de los múltiples parámetros de configuración en el resultado del mismo, etc ...para más datos en este aspecto se puede consultar el proyecto de fin de carrera ***"Planificació de trajectòries de dues dimensions per a robots mòbils"*** de Joaquim Galceran Capeta, en el que esta basado el algoritmo de "path planning". Sin embargo se hará especial hincapié en los aspectos que fueron modificados con respecto al algoritmo original, así como los resultados obtenidos.

7.6.1. DETERMINACIÓN DE LOS PUNTOS DE PARTIDA Y DESTINO

Como ya se comentó en capítulos anteriores (ver apartado 6.2.1. *Construcción del mapa de trayectorias*), el algoritmo de "path planning" original no consideraba el punto de partida y de destino como puntos de la malla de posibles trayectorias, escogiendo en su lugar los más cercanos de los que se habían lanzado aleatoriamente. Por los motivos que se abdujeron en el mencionado capítulo, ese defecto fue subsanado haciendo que efectivamente, tanto el punto de partida como el de destino fuesen la posición del centro del robot y el punto indicado por el usuario respectivamente.

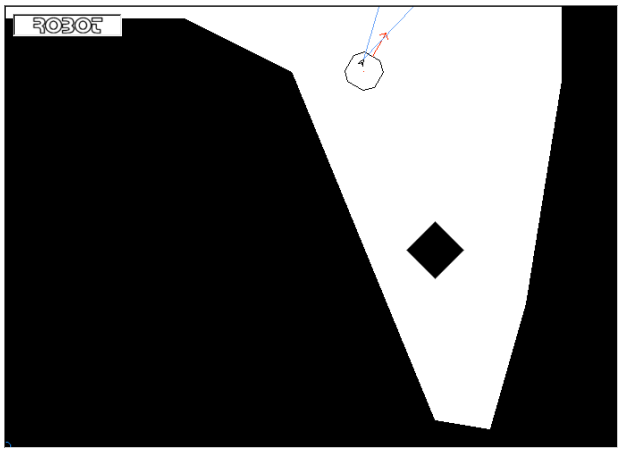


Figura 7.8: Imagen del mapa utilizado en el experimento tal y como queda representado en un primer momento

Supóngase la carga del mapa predefinido de la *Figura 7.8*. Para verificar la corrección de los puntos observados como inicio y final de trayectoria, se someterá el mapa a todas las transformaciones que permite la aplicación, es decir, rotación, escalado y traslación del mapa antes de indicarle al programa cual es el punto de destino. Una vez realizadas todas estas operaciones, el mapa queda de la siguiente manera:

En la *Figura 7.9* se muestra el mapa de la *Figura 7.8* convenientemente escalado, rotado y trasladado, para el experimento. Ahora que puede verse todo el mapa, se define como punto de destino el que se encuentra marcado por la cruz del cursor, en medio de la habitación cuadrada de la imagen.

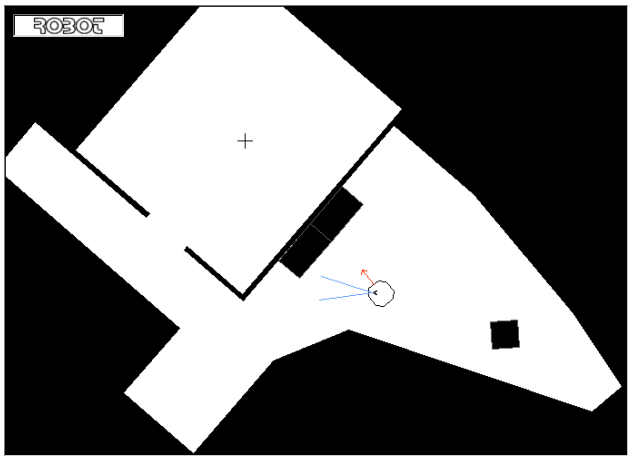


Figura 7.9: Imagen del mapa después de escalarlo, rotarlo y desplazarlo

Fijando los parámetros del "path planning" a 50 puntos como máximo, conexión punto por punto, distancia máxima de conexión a 2000 mm, distancia mínima de conexión a 2 mm, K vecinos a 6, número máximo de conexiones por nodo a 4 y definiendo una ventana de resolución del 70 %, el programa arroja los resultados de la *Figura 7.10*.

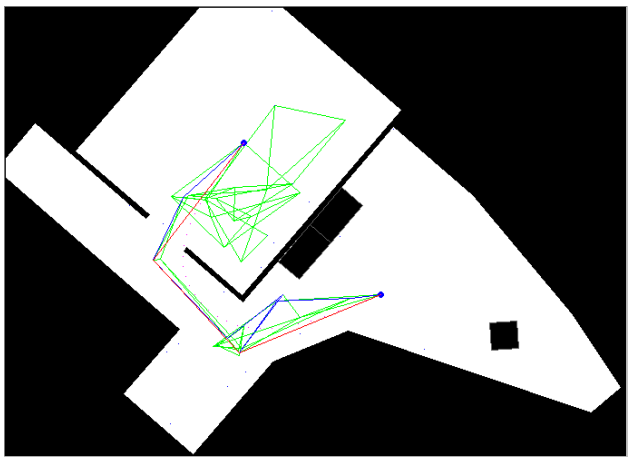


Figura 7.10: Cálculos realizados por el programa para determinar la trayectoria entre la posición actual del robot y el punto de destino indicado.

Sin entrar en consideraciones tales como si el camino encontrado es el óptimo, si se superponen las imágenes de las *Figuras 7.9 y 7.10* se puede observar como, incluso después de todas las trasformaciones que ha sufrido el mapa de mundo, tanto el punto de origen como el de destino coinciden con la posición del robot y con el punto determinado por el cursor respectivamente (ver *Figura 7.11*, página siguiente).

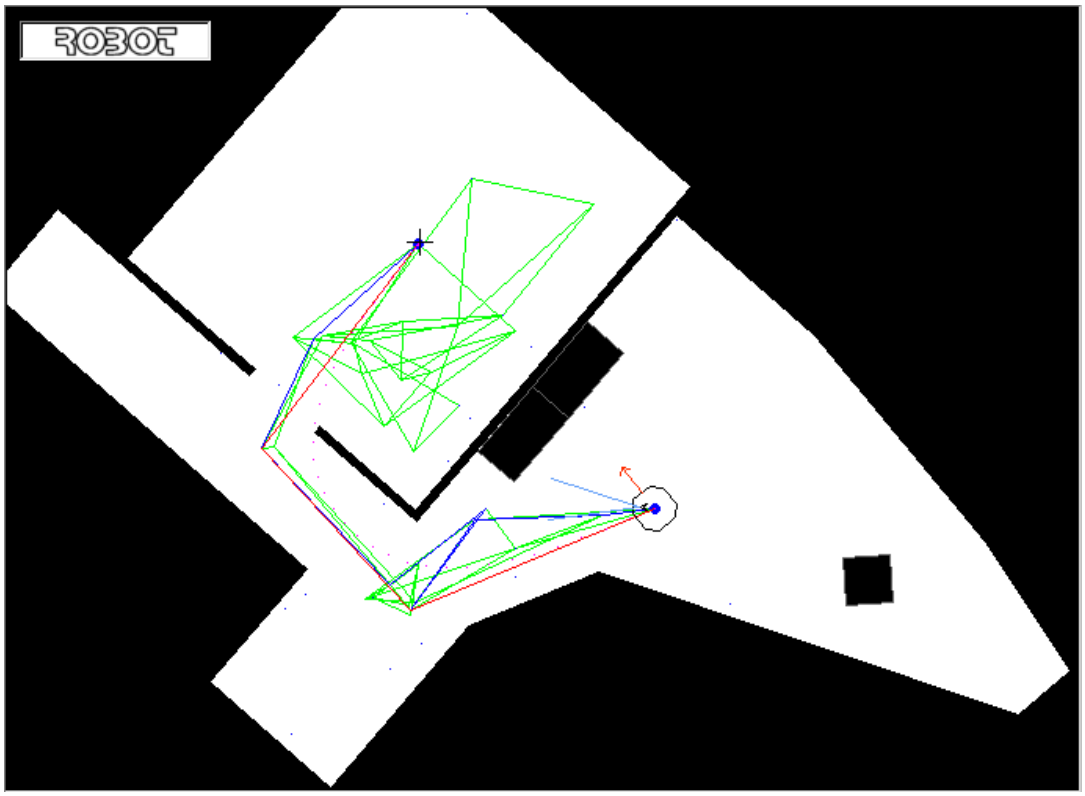


Figura 7.11: Superposición de las imágenes correspondientes a las *Figuras 7.9* y *7.10*. Como se puede observar, tanto el robot como el cursor que marcaba el punto de destino en la *Figura 7.9*, coinciden perfectamente con sus homónimos de la *Figura 7.10*

7.6.2. CONEXIÓN ENTRE NODOS DURANTE LA GENERACIÓN DEL MAPA DE CAMINOS

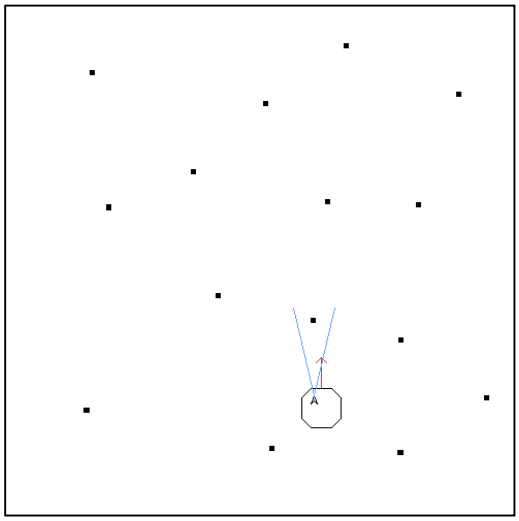


Figura 7.12: Mapa utilizado en el experimento. Vemos claramente la diferencia de tamaño entre el robot y los objetos del mapa

A continuación se demuestran los resultados obtenidos por el nuevo método utilizado para verificar la conexión entre nodos, durante la fase de creación del mapa de caminos. A tal fin, se ha decidido cargar un mapa pregenerado repleto de objetos de un tamaño aproximadamente 10 veces inferior al robot (ver *Figura7.12*), que muy bien podrían ser patas de mesas, sillas, u otros objetos similares. A continuación se le ordena al robot que genere el mapa de trayectorias.

Para la prueba, el número de puntos lanzados han sido 200, con conexión punto por punto, una distancia máxima de conexión entre nodos de 1 metro, sin ventanas de resolución, y con un radio de seguridad del robot de 300 mm.

Después de numerosas pruebas, en ninguno de los casos se han unido dos puntos que estuviesen demasiado cerca de un objeto del plano. En la *Figura 7.13* podemos ver el resultado de una de las numerosas pruebas realizadas.

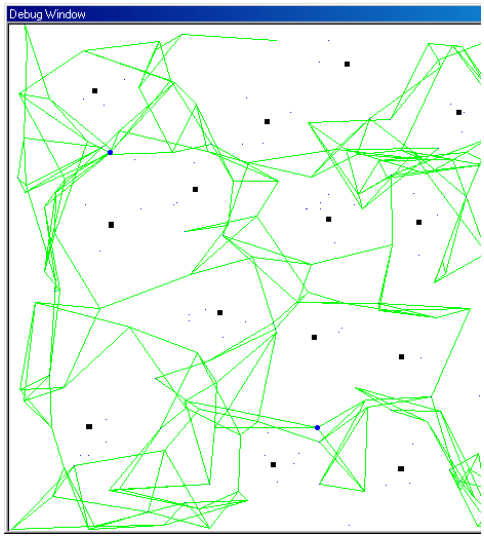


Figura 7.13: Ejemplo de generación de mapa de caminos

En conclusión, aunque seguramente los sonares no llegarían a detectar objetos tan pequeños (estos tendrían que ser detectados mediante estéreo-visión), podemos estar seguros que si representamos un objeto en el mapa, por pequeño que sea, el algoritmo de generación de trayectorias jamás trazaría una que colisionase con él.

8 CONCLUSIONES

8.1 IMPLEMENTACIÓN

Para implementar el Navegador han sido necesarios 2 entornos de programación.

Por un lado se ha hecho uso del entorno de programación Visual C++ 6.0 para el desarrollo de las funciones encargadas de controlar el robot, el Pan&Tilt, hacer muestreo de resultados, etc... . Todas estas funciones han sido compiladas y agrupadas en una DLL (Dynamic Link Library) llamada DllControl.dll, con el fin que puedan ser utilizadas en otras implementaciones. El desarrollo del Pan&Tilt Conection Listener (PTCL) también se realizó en este entorno de programación

Por otro lado, fue necesario el uso del entorno de programación Visual Basic 6.0 para el desarrollo de la interfaz de usuario del Navegador, encargada de recibir las ordenes del usuario, interpretarlas, y llamar a las funciones pertinentes en DllControl.

8.2 RESULTADOS

A lo largo del capítulo 6 se muestran los resultados obtenidos durante la evaluación de los diferentes aspectos de la interfaz.

Puede decirse que el Navegador permite al usuario controlar el robot, su cabeza y sensores, así como obtener información de todos los aspectos relativos al control, en tiempo real.

La generación de trayectorias ha perdido velocidad con respecto a la implementación original, pero a ganado en fiabilidad.

El uso de ordenadores más potentes que el especificado en apartado 3.4.1.2. del capítulo de Descripción Hardware y Software, permitiría unos tiempos de respuesta del robot y Pan&Tilt menores, y una tasa de refresco de imágenes mayor.

8.3 PROPUESTAS DE MEJORA

Aunque los objetivos iniciales del proyecto han sido alcanzados, podrían considerarse las siguientes mejoras:

- Representación de la información del mundo y los objetos en 3D (OpenGL), permitiendo distintas vistas, zooms y rotaciones.
- La información del entorno extraída mediante estéreo visión, debería integrarse en el mapa del mundo junto con la información de los sonares.
- Incorporar la función de generar un mapa del entorno del robot, a partir de la información de los sensores.

BIBLIOGRAFÍA

(FGL93) Fu K., Gonzales R., Lee C., *Robotics - Control, Sensing, Vision and Intelligence*, McGraw-Hill International Editions, 1993

(KoM 96) Konolige K., Myers K., *The Saphira Architecture for Autonomous Mobile Robots*, Documento en formato PDF, Noviembre de 1996.

(SRK 93) Saffiotti A., Ruspini E. H., Konolige K., *Blending reactivity and goal-directedness in a fuzzy controller*, Proc. IEEE Int. Conference on Fuzzy Systems, páginas 134-139, San Francisco, CA 1993.

(KoK 98) Konolige K., *Saphira Software Manual ver6.1e*, Documento en formato PDF, Abril de 1998

(MIU 98) *Matrox Imaging Library ver5.1, User Guide,* Documento en formato PDF, Febrero de 1998.

(MIB 98) *Matrox Imaging Library ver5.1, Board-specificNotes,* Documento en formato PDF, Febrero de 1998.

(MIC 98) *Matrox Imaging Library ver5.1, Command Reference,* Documento en formato PDF, Febrero de 1998.

(GaJ 99) Galcerán J. , *Planificació de trajectòries de dues dimensions per a robots mòbils*, Proyecto Final de Carrera, 1999.

(PFT 93) Press W., Flannery B., Teukolsky S., Vetterling W., *Numerical Recipes in C*, Cambridge University Press, páginas 113-116, Febrero de 1993

(WeK 94) Weiler K., *An Incremental Angle Point in Polygon Test*, Graphics Gems IV, páginas 16-23, Academic Press, 1994

(WkM 91) W. Kernighan, B.; M. Ritchie D.; *El lenguaje de programación C*, Prentice-hall hispanoamericana S.A., Segunda Edición, 1991

(LeA 99) Leinecker R., Archer T., *Visual C++ 6*, Anaya Multimedia, 1999

(CeF 99) Ceballos F., *Enciclopedia de Microsoft Visual Basic 6*, Ra-Ma 1999

(CVo) Página HTML: *CVonline: Vision Geometry and Mathematics,* http://www.dai.ed.ac.uk/CVonline/geom.html

(YTT) Yoshiyuki T., Takashi K., Tateki U., Masao M., Hiroyuki K., Hiroyasu F., *Development of the Mobile Robot System to aid the daily life for physically handicapped(Interface using internet browser)*, Página HTML: http://www.dinf.org/tide98/19/takahashi_yoshiyuki.html

(Rob) Página HTML: *Robot Controller Project*, http://www.ee.byu.edu/ee/srprojects/robot/1998/shakey/software/

(CaS) Página HTML: *The CareBot Software*, http://www.geckosystems.com/carebot/robotsoftware.html

ANEXO 1: PRESUPUESTO

A1.1 DATOS DE PARTIDA

En este primer apartado se especifican los costes por hora establecidos para la confección del presupuesto: por un lado, el coste por hora del personal, y por otro, el coste por hora de utilización de equipos.

A1.1.1 COSTE POR HORA DEL PERSONAL

Se estima que el coste por hora del personal dedicado al desarrollo del proyecto es el de la siguiente tabla:

Analista	7000 ptas/hora
Programador	5000 ptas/hora
Operador	3000 ptas/hora

A1.1.2 COSTE POR AÑO DE UTILIZACIÓN DE LOS EQUIPOS

A fin de calcular el coste por año de utilización de los equipos, debe tenerse en cuenta: el coste del personal del Centro de Cálculo (CCIRI) del IRI (Instituto de Robótica y Informática Industrial), los contratos de mantenimiento y, finalmente, la amortización de los equipos.

A1.1.2.1 COSTE DEL EQUIPO DE NAVEGACIÓN

El equipo de control de navegación junto con el robot en si, está formado por:

- Robot Pioneer 2-DX. Su valor se estima en 1.740.000 ptas

- Ordenador PC Pentium III 600 MHz, 128MB de RAM, 10 GB de disco duro (HD) y monitor de 17''. Su precio estimado es de 200.000 ptas.

- Tarjeta de adquisición y procesado de imágenes Matrox METEOR. Su precio se estima en 150.000 ptas.

- 2 cámaras digitales color, modelos SONY EVI-371DG. Su precio por unidad se estima en 135.000 ptas.

- 2 Video Senders. Su precio estimado en conjunto es de 230.000 ptas

- Radio Ethernet modelo AirEZY2400. Su precio por unidad se estima en 125.000 ptas

- Sistema de orientación de las cámaras (Pan&Tilt) modelo PTU-46-17.5. Su precio estimado es de 450.000 ptas.

- Software necesario: librerías MIL (Matrox Imaging Library). Su precio se estima en 470.000 ptas

Por tanto el coste total del equipo de navegación es:

Robot	1.740.000
Ordenador	200.000
Tarjeta Matrox METEOR	150.000
Cámaras	2 X 135.000 = 270.000
Video Senders	230.000
Ethernets	2 X 125.000 = 250.000
Pan&Tilt	450.000
Software	470.000
Coste total del equipo de navegación	3.770.000 ptas

Considerando la amortización del equipo en 5 años, con unos intereses del 15%, la anualidad de la amortización es:

A1.1.2.2 COSTE DE MANTENIMIENTO

$$COSTE(Equipo) = 3.770.000 \times \frac{0.15 \times (1.15)^5}{(1.15)^5 - 1} = 1.124.650\, ptas/año$$

El coste de mantenimiento anual del equipo se estima en un 10% del valor de total de compra, por tanto:

$$COSTE(Mantenimiento) = 3.770.000 \times 0.1 = 377.000 ptas/año$$

A1.1.2.3 GASTOS VARIOS

Se incluye el consumo eléctrico y posibles reparaciones

$$COSTE(Varios) = 35.000 ptas/año$$

A1.1.2.4 GASTOS DE PERSONAL

Se consideran los costes de un operador trabajando durante 5 horas/semana, 40 semanas al año

$$COSTE(Personal) = 3000 \times 5 \times 40 = 600.000 ptas/año$$

A1.1.2.5 COSTE POR AÑO DE UTILIZACIÓN DE LOS EQUIPOS

Teniendo en cuenta los costes anteriores, se calcula un coste anual de utilización de los equipos de:

Coste del equipo de navegación	1.124.000 ptas/año
Coste de mantenimiento	377.000 ptas/año
Gastos varios	35.000 ptas/año
Gastos de personal	600.000 ptas/año
Coste total anual de utilización de los equipos	**2.136.000 ptas/año**

Si se estima que se trabajan 50 semanas al año, un total de 40 horas semanales y con un factor de utilización del equipo de un 80 %, finalmente se obtiene:

Coste total por hora de utilización de los equipos = 1.335 ptas/hora

A1.2 COSTE DE DESARROLLO

Teniendo en cuenta los precios del apartado anterior, se calculará el coste de desarrollo del proyecto, equipo, y gastos varios. No se tendrá en cuenta el coste del inmovilizado del IRI (Instituto de Robótica e Informática Industrial), sino solo los de desarrollo del proyecto en referencia a los recursos humanos y equipos informáticos.

A1.2.1 COSTE DEL PERSONAL

Aproximadamente, se estima que el tiempo empleado para el desarrollo del este proyecto ha sido de 75 semanas (un año y medio trabajando 50 semanas al año), con una dedicación media de 40 horas semanales. El tiempo empleado puede ser dividido en: un 20% del tiempo como analista, un 70% como programador y un 10% como operador.

Coste (Analista)	600 h x 7000ptas/h =	4.200.000 ptas
Coste (Programador)	2100 h x 5000ptas/h=	10.500.000 ptas
Coste (Operador)	300 h x 3000ptas/h=	900.000 ptas
Coste (Personal)		**15.600.000 ptas**

A1.2.2 COSTE DE UTILIZACIÓN DE LOS EQUIPOS

Del total del tiempo dedicado, un 80% ha sido a la implementación de programas y utilización del equipo de Navegación, por tanto:

Tiempo de utilización del equipo: 75 semanas X 40 horas/semanales X 0,8 = 2400 h

Coste de utilización de los equipos = 2400 h x 1335 ptas/hora = 3.204.600 ptas

A1.2.3 GASTOS VARIOS

Se consideran los gastos asociados a los siguientes conceptos:

Documentación	30.000 ptas
Papel e impresión	20.000 ptas
Gastos varios	**50.000 ptas**

A1.2.4 COSTE TOTAL DE DESARROLLO

El coste total del desarrollo del proyecto es el resultado de sumar todos los costes anteriores, por lo tanto:

Coste (Personal)	15.600.000 ptas
Coste (Equipo de Navegación)	3.204.600 ptas
Gastos Varios	50.000 ptas
Coste (TOTAL)	**18.854.600 ptas**

ANEXO 2: MANUAL DE USUARIO

A2.1. DESCRIPCIÓN DE LA INTERFAZ

Una vez instalado el programa, durante su ejecución aparece una ventana de bienvenida como la de la *Figura 2.1*

Figura 2.1: Ventana de presentación del navegador

En la zona inferior de la ventana de presentación se puede observar una pequeño cuadro de diálogo que informa como el programa esta intentando determinar si será capaz de utilizar DirectInput (parte del DirectX de Windows encargada de los dispositivos de entrada al sistema) o no. En la mayoría de los casos la respuesta será afirmativa, a no ser que la aplicación este siendo ejecutada en un ordenador con sistema operativo Windows NT, en cuyo caso la respuesta será negativa. Una vez detectado o no el DirectInput, la ventana permanecerá 3 segundos en pantalla si no se hace click sobre ella con el botón izquierdo del ratón.

La pantalla que aparece a continuación es la pantalla principal del Navegador (ver *Figura 2.2*), desde la que se puede tener acceso a todos los controles del mismo.

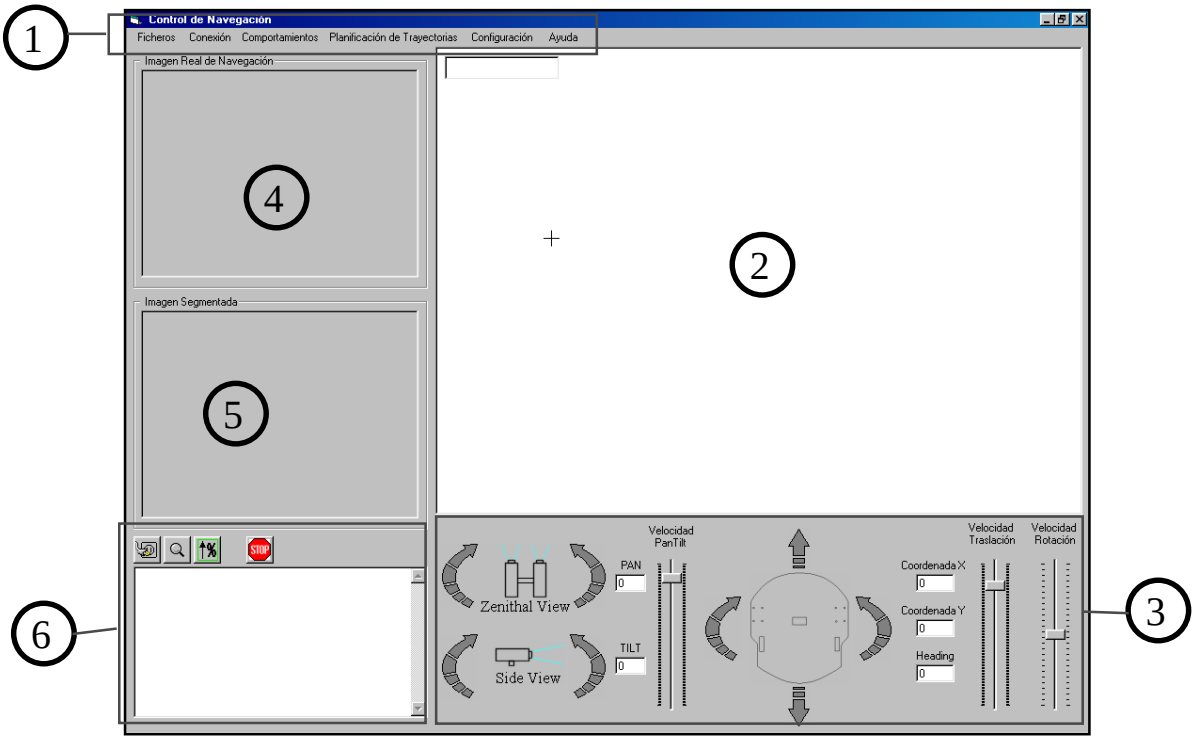


Figura 2.2: Pantalla principal del navegador

Pueden distinguirse las siguientes zonas:

1. Barra de menús
2. Mapa de navegación
3. Controles de navegación
4. Imagen de la cámara de navegación
5. Imagen segmentada
6. Botonera y cuadro de mensajes

A2.1.1 BARRA DE MENÚS

Los barra de menús está dividida en los siguientes apartados:

A2.1.1.1. FICHEROS

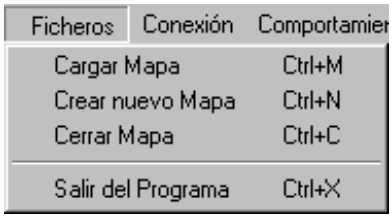


Figura 2.3: Menú Ficheros

Este menú da acceso a las principales acciones relacionadas con el mapa, tales como cargar un fichero con un mapa predefinido por el usuario (Ctrl.+M), comenzar la creación de un mapa con un mapa en blanco (Ctrl.+N) o cerrar el mapa que este cargado en la aplicación en ese momento (Ctrl.+C).

Como sucede en casi todos los programas, también se dispone de la opción de salir del programa.

Nótese que, en este caso, todas las opciones de este menú disponen de una combinación de teclas de acceso directo. Esto es así porque son acciones muy comunes y es conveniente poder acceder a ellas sin necesidad de dirigirse a los menús.

A2.1.1.2. CONEXIÓN

Este menú permite al usuario conectarse o desconectarse del robot (simulador o robot real, como ya se verá en el apartado de configuración), del Pan&Tilt, o de ambos al mismo tiempo.

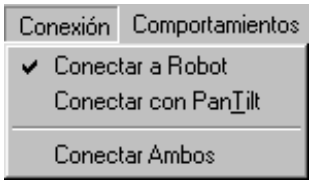


Figura 2.4: Menu comportamientos

Como se puede apreciar en la *Figura 2.4*, cuando se ha realizado la conexión con uno de ellos aparece un símbolo que nos indica que la conexión ya ha sido realizada. En ese caso volver a hacer click con el ratón provocaría la desconexión.

La razón por la que es posible conectar y desconectar independientemente el robot y el Pan&Tilt es que, en algunas ocasiones, cuando se intenta conectar con ambos a la vez, el programa logra realizar la conexión solo con uno de los dos. De esta manera, es posible volver a intentar la conexión con el dispositivo que falló sin necesidad de desconectarse del que ya lo está. También es posible conectarse exclusivamente con el Pan&Tilt para hacer pruebas de estéreo-visión, sin conectarse al robot.

En este caso no se dispone de teclas de acceso directo ya que, aunque son acciones muy comunes, existe un botón en el panel de control que permite conectarse a ambos dispositivos con un solo click de ratón (ver apartado.*1.6 Botonera y cuadro de mensajes*)

A2.1.1.3. COMPORTAMIENTOS

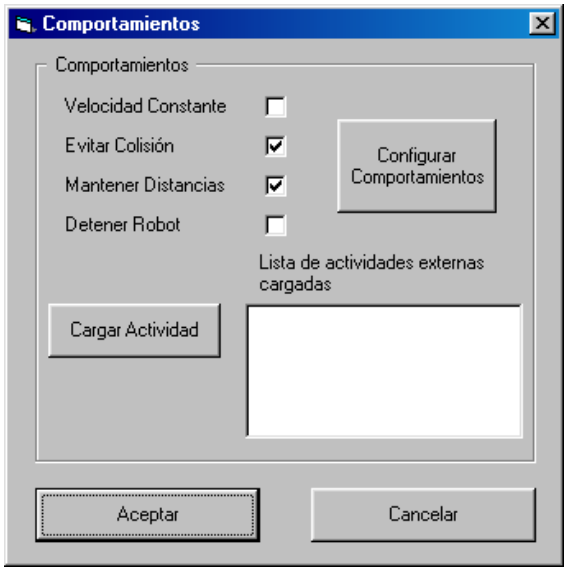


Figura 2.5: Ventana de control de comportamientos

Haciendo un click con el ratón en esta opción de la barra de menús, se abre inmediatamente la ventana de control de comportamientos(ver *Figura 2.5*).

En esta ventana es posible activar y desactivar los comportamientos predefinidos de Saphira, así como cargar actividades externas diseñadas por el usuario.

A2.1.1.4. PLANIFICACIÓN DE TRAYECTORIAS

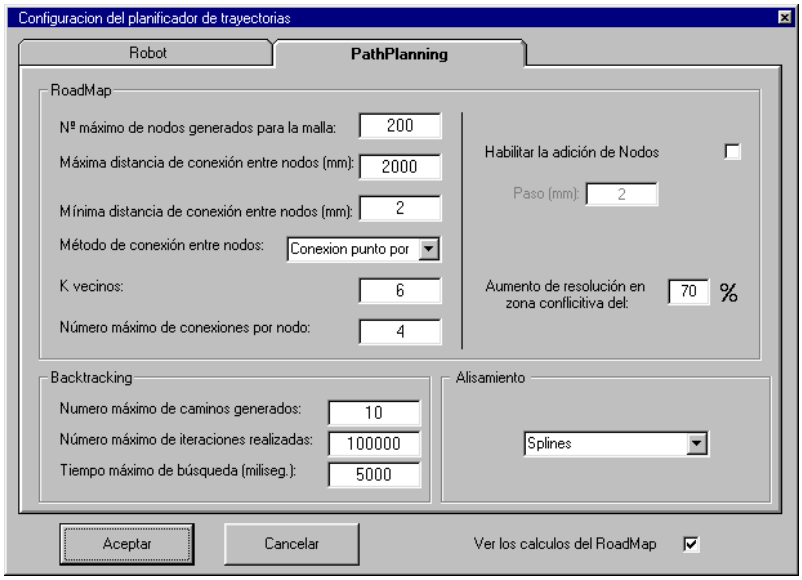


Figura 2.6: Ventana de configuración de planificación de trayectorias.

Esta opción de la barra de menús abre la ventana de configuración del planificador de trayectorias. La configuración de los parámetros necesarios para la planificación de trayectorias será tratada más adelante en el apartado 2.2.2. de este mismo Anexo.

A2.1.1.5. CONFIGURACIÓN

La ventana que se abre en esta ocasión es la ventana de configuración del propio navegador.

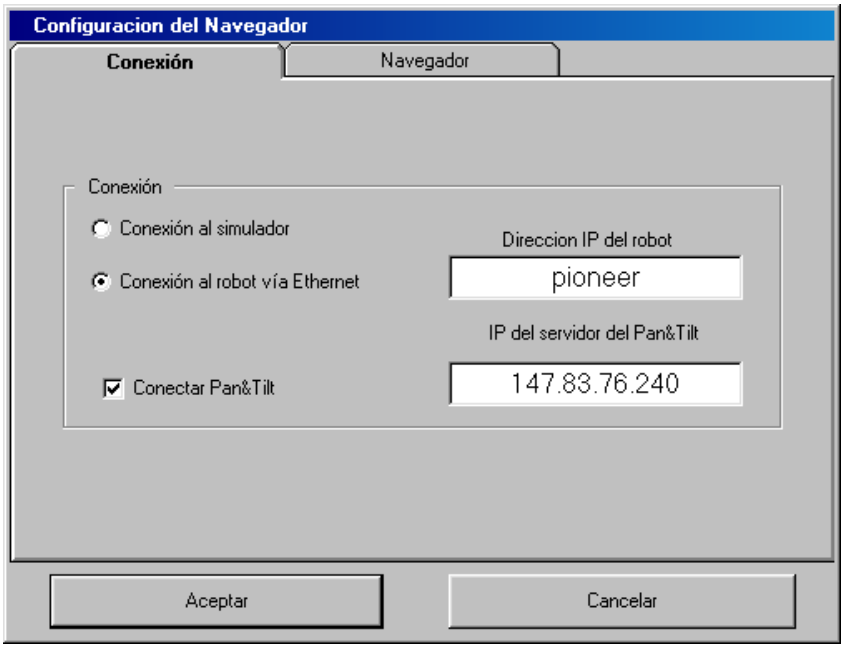


Figura 2.7: Ventana de configuración del Navegador

La configuración del navegador se encuentra dividida en dos secciones, la configuración de la conexión con Robot y Pan&Tilt, y la configuración de los parámetros de representación del plano de navegación

A2.1.2. MAPA DE NAVEGACIÓN

Esta es quizás la zona más importante del Navegador. En ella es donde se representan las evoluciones del robot así como la información de los sonares, permitiendo su seguimiento. En la esquina superior izquierda del mapa de navegación, puede apreciarse un pequeño recuadro blanco (ver *Figura 2.8* ,página siguiente) que informa del dispositivo que se está controlando en ese instante (robot o Pan&Tilt).

Existen dos posibles modos de visualización del mapa. El mapa centrado en el mundo, es decir, el mundo permanece fijo mientras que es el robot el que se desplaza. La segunda posibilidad es la representación del mapa centrado en el robot, es decir, el robot permanece fijo el centro del mapa de navegación y es el mundo el que se mueve a su alrededor.

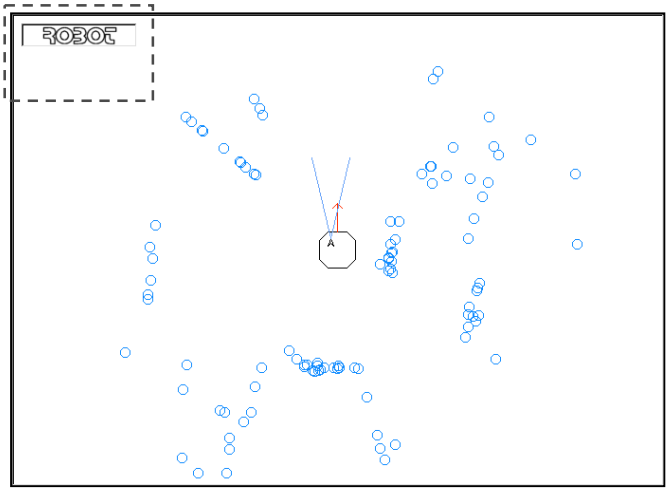


Figura 2.8: Mapa de navegación. El marco punteado de la imagen señala el recuadro que informa del dispositivo que se esta controlando en cada momento

A2.1.3. CONTROLES DE NAVEGACIÓN

Los controles de navegación ponen a disposición del usuario todos los datos necesarios para la navegación guiada del robot.

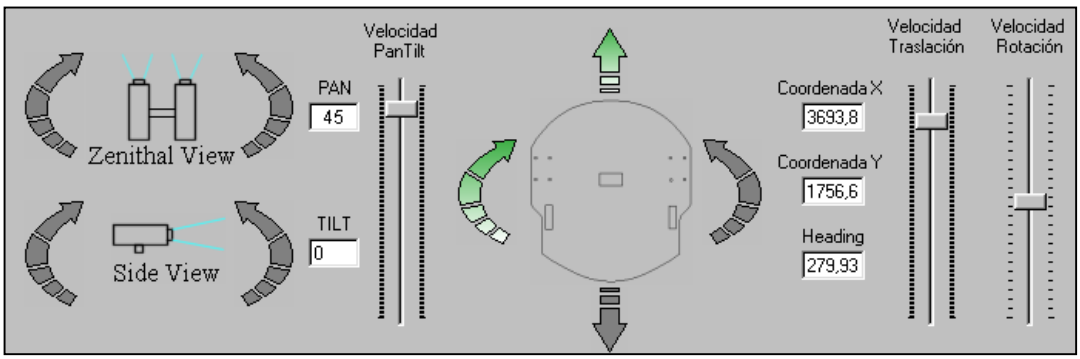


Figura 2.9: Controles de navegación

Podrían distinguirse dos zonas: por un lado los controles de navegación del Pan&Tilt, que permiten controlar la velocidad de giro de sus ejes y comunican la orientación de estos; y por otro los controles de navegación del robot, que dan la posición (en milímetros respecto al origen de coordenadas del mapa) y orientación del robot, además de permitir un control independiente de la velocidad de traslación y rotación.

Para evitar que el usuario pueda perder el sentido de la orientación de los movimientos que le ordena al robot, en el cuadro de control se iluminan unas flechas que ilustran claramente la dirección y sentido del movimiento que se está realizando en cada instante.

A2.1.4. IMAGEN DE LA CÁMARA DE NAVEGACIÓN

Esta ventana del Navegador proporciona imágenes de lo que el robot "ve", siempre y cuando las cámaras se encuentren conectadas. Su frecuencia de actualización es variable, y puede configurarse entre 5 y 60 FPS (frames por segundo).

A2.1.5. IMAGEN SEGMENTADA

Esta capacidad no esta implementada todavía. En un futuro permitirá recibir imágenes de la cámara de navegación sobre las que se irán señalando los objetos reconocidos por el robot.

A2.1.6 BOTONERA Y CUADRO DE MENSAJES

La botonera permite tener acceso a funciones del Navegador de otra manera inaccesibles, o simplemente ofrece una ejecución rápida de funciones muy comunes.

El cuadro de mensajes es, como su propio nombre indica, la ventana donde el navegador informa al usuario de sus acciones o de la ejecución de sus ordenes.

A2.2 CONFIGURACIÓN

A2.2.1. CONFIGURACIÓN DEL NAVEGADOR

Como ya se comentó brevemente en la descripción de la interfaz, existen dos partes bien diferenciadas en la ventana de configuración del navegador: configuración de la conexión del Navegador con otros dispositivos (pestaña **Conexión**), y configuración de la interacción usuario-navegador (pestaña **Navegador**).

A2.2.1.1. CONFIGURACIÓN DE LA CONEXIÓN

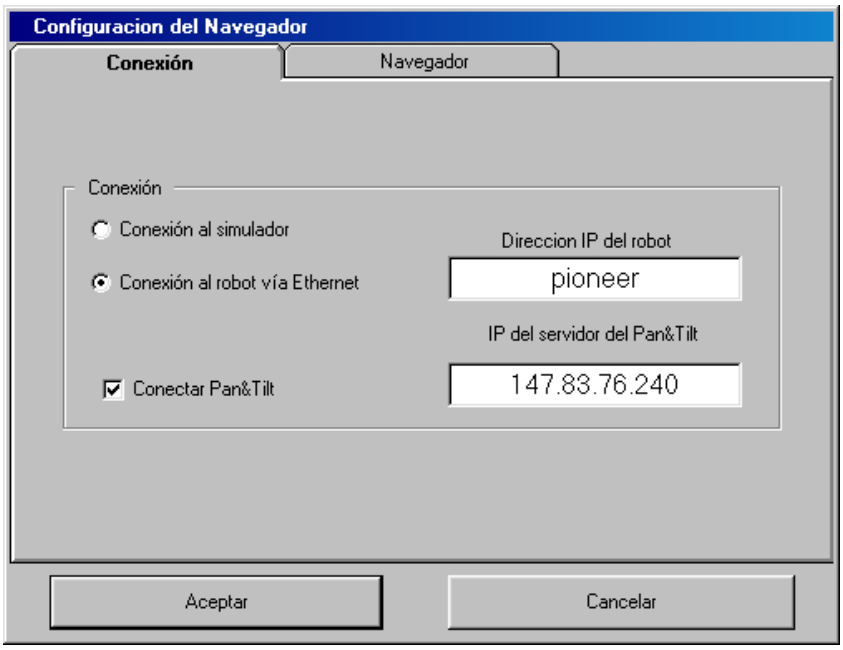


Figura 2.10: Ventana de configuración del Navegador. Pestaña de configuración de la conexión.

El Navegador permite al usuario conectarse al simulador que Saphira suministra junto con el software del robot, o conectarse al robot real vía Ethernet, en cuyo caso es necesario proporcionarle al programa la dirección IP del PC de a bordo del robot, en el que deberá estar funcionando el IPTHRU.

La opción "Conectar Pan&Tilt" determina si se ha de establecer, o no, conexión con el Pan&Tilt situado en la cabeza del robot. Al igual que en el caso de la conexión con el robot real, será necesario facilitarle al navegador la dirección IP del PC de abordo del robot, en el que necesariamente tiene que estar instalado y funcionando el programa que acompaña a este navegador (Pan&Tilt Conection Listener o PTCL). Normalmente es recomendable no dejar esta opción activada a no ser que se vaya a realizar la conexión con el Pan&Tilt. Esto se debe a que el tiempo que implica intentar la conexión con el Pan&Tilt cuando este no existe, hace recomendable deshabilitar la opción en lugar de permitir que de una conexión errónea.

ATENCIÓN: La conexión con el Pan&Tilt implica la conexión con la cámara de navegación. Si no se dispone de una tarjeta Matrox METEOR instalada en el PC, es altamente recomendable no realizar la conexión con el Pan&Tilt, ya que de otro modo el muy probable que la aplicación termine de forma inadecuada.

A2.2.1.2. CONFIGURACIÓN DEL MAPA DE NAVEGACIÓN

Figura 2.11: Ventana de configuración del Navegador. Pestaña de configuración del mapa de navegación.

En esta pestaña se encuentran las opciones que permiten determinar la manera en que el navegador interactúa con el usuario.

La representación del mapa de navegación es posible configurarla de dos maneras distintas: centrada en el mundo, o centrada en el robot. En la representación centrada en el mundo, este se mantiene fijo y es el robot el que se desplaza en mapa. Esta opción permite al usuario la posibilidad de rotar, escalar y desplazar el mapa a su antojo. En la representación centrada en el robot, es este último el que se mantiene fijo en el centro de la ventana con su dirección de avance orientada según el sentido positivo del eje Y, siendo el mundo el que se mueve a su alrededor. En este caso no es posible trasladar ni rotar el mapa, pero sí escalarlo.

También pueden encontrarse aquí las opciones de control: Teclado o Joystick. Si al seleccionar la opción de control por Joystick, no es posible controlar el robot, compruébese que el Joystick esta bien conectado al sistema y es detectado por este. Si los problemas persisten, seleccionar control por teclado.

A la derecha del cuadro de configuración de representación del mapa, 3 deslizadores permiten fijar (mencionados de derecha a izquierda), la frecuencia de refresco de la cámara de navegación, la frecuencia de control del Pan&Tilt, y la frecuencia de control del robot. Los valores que toman estas variables, aparecen en pantalla al mantener pulsado el botón izquierdo del ratón sobre cada uno de los deslizadores.

La frecuencia de refresco de la cámara de navegación, medida en frames por segundo (a partir de este momento se hará referencia como FPS), determina el número de imágenes por segundo que se adquieren de la cámara de navegación. Esta frecuencia puede variar entre 5 y 60 FPS, siendo más suave la animación cuanto más alto sea este número, aunque normalmente no es necesario subirlo por encima de 30 FPS.

La frecuencia de control del Pan&Tilt determina el número de veces por segundo que el Navegador consulta el estado del dispositivo de control (teclado o Joystick). En este caso si que es conveniente fijar esta cifra al más alto valor que permita el sistema, ya que esto proporcionará un control mucho más preciso sobre el Pan&Tilt.

Al igual que en el caso del Pan&Tilt, la frecuencia de control del robot establece el número de consultas por segundo que el Navegador realiza al dispositivo de control (teclado o Joystick). También en este caso es muy recomendable establecer la frecuencia de control del robot lo más alta posible, ya que esto se traduciría en una respuesta más rápida del robot.

NOTA: El peligro de fijar estas frecuencias muy altas, sobre todo en el caso del control del robot, es sobrecargar de trabajo el sistema, observando como a este le cuesta abrir menús, cambiar de aplicación, e incluso completar las tareas del propio Navegador. Por ese motivo, si se observa una respuesta del sistema anormalmente lenta, es muy recomendable disminuir en la medida de los posible estos valores.

A2.2.2. CONFIGURACIÓN DEL GENERADOR DE TRAYECTORIAS

La configuración del planificador de trayectorias dispone de dos pestañas. En la pestaña "Robot" se establece el radio de seguridad (en milímetros) que empleará el generador de trayectorias para definir las zonas seguras para el robot. En la segunda pestaña, la que aparece por defecto cuando se abre la ventana, se encuentran todos los parámetros configurables del planificador de trayectorias, agrupados según la parte del proceso de generación al que afecte.

A2.2.2.1. CONFIGURACIÓN DEL ROADMAP

- **Número máximo de nodos generados para la malla**: Determina la cantidad máxima de nodos que se generan al azar para trazar el mallado.

- **Máxima distancia de conexión entre nodos:** Distancia máxima (medida en milímetros) a la que se considera la posibilidad de unir dos nodos cualesquiera.

- **Mínima distancia de conexión entre nodos:** Distancia mínima (medida en milímetros) a la que se considera la posibilidad de unir dos puntos cualesquiera.

- **Método de conexión entre nodos:** Dos opciones disponibles: Conexión punto por punto, y Pregeneración de puntos. En el primer caso a medida que

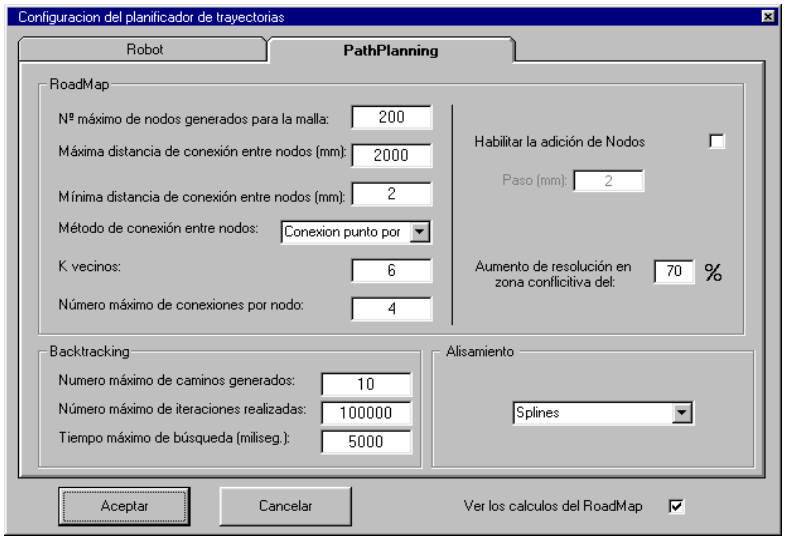


Figura 2.12: Ventana de configuración de planificación de trayectorias.

se van generando los puntos del mallado, el algoritmo trata de unirlos con los generados previamente. La pregeneración de puntos, como su propio nombre indica, implica generar la totalidad de puntos del mallado antes de intentar cualquier tipo de conexión entre ellos.

- **K vecinos:** Establece el número de vecinos a los que se intentará conectar cada nodo.

- **Número máximo de conexiones por nodo:** Número máximo de conexiones permitidas por nodo.

- **Habilitar la adición de nodos:** Habilita la adición de una red de nodos según el paso especificado en milímetros.

- **Aumento de resolución en zona conflictiva:** Determina el porcentaje de puntos generados al azar que deben estar en la ventana de resolución definida por el usuario.

A2.2.2.2. CONFIGURACIÓN DE LA BÚSQUEDA DEL CAMINO MÁS CORTO (BACTRACKING)

- **Número máximo de caminos generados:** El Navegador dispone de un contador interno que determina la cantidad de caminos encontrados que unen el nodo de salida con el de llegada. El número máximo de caminos generados fija la cantidad de estos caminos que debe encontrar el navegador antes de dar por finalizado el proceso.

- **Número máximo de iteraciones realizadas:** Este parámetro es un parámetro de seguridad que impide que el navegador quede bloqueado en casos de mallados complejos.

- **Tiempo máximo de búsqueda:** Limita el tiempo, expresado en milisegundos, que dispone el Navegador para realizar la búsqueda del camino más corto. Cuando el reloj interno marca el tiempo especificado por el usuario, el Navegador escogerá el camino más corto que halla encontrado hasta el momento. La limitación temporal que establece el usuario solo abarca el proceso de búsqueda del camino más corto, sin incluir la generación del mallado.

A2.2.2.3. CONFIGURACIÓN DEL ALISAMIENTO

Se puede determinar el tipo de alisamiento que se desea aplicar a la trayectoria escogida por el Navegador, de entre las dos disponibles:

- **Splines:** Es el método predeterminado. Traza una curva que no pasa sobre los puntos que determinan la trayectoria del robot, a excepción de los puntos de salida y llegada..

- **Radio de curvatura:** Alisa las esquinas de la trayectoria escogida por el Navegador insertando un arco de circunferencia tangente a los tramos rectilíneos de la misma.

A2.2.3. CONFIGURACIÓN DE LOS COMPORTAMIENTOS PREDEFINIDOS

Para acceder a la ventana de configuración de comportamientos es necesario pulsar el botón *Configurar Comportamiento*, dentro de la ventana de control de comportamientos (ver de nuevo el apartado 1.2.3. de este mismo Anexo). La nueva ventana que se abre contiene una pestaña por cada comportamiento configurable.

A2.2.3.1. VELOCIDAD CONSTANTE

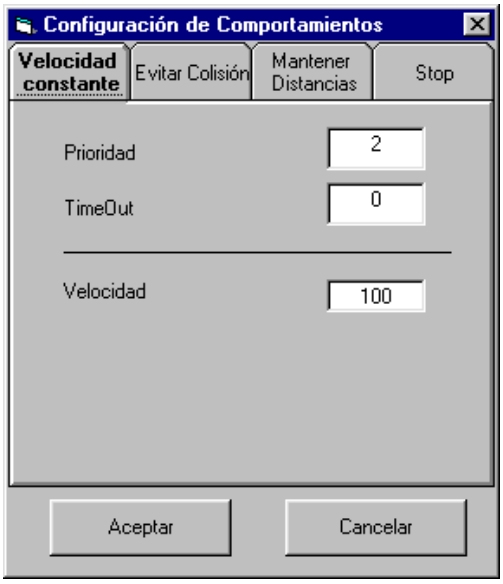


Figura 2.13: Pestaña de configuración del comportamiento Velocidad Constante

- **Prioridad**: Prioridad del comportamiento frente a los demás. A menor número, mayor prioridad. Valor predeterminado 2.
- **Timeout:** Tiempo expresado en milisegundos que permanecerá activo el comportamiento. Valor predeterminado 0, cuyo significado es que permanecerá activo indefinidamente.
- **Velocidad:** Velocidad expresada en milímetros por segundo que se desea que el robot mantenga constante. Valor predeterminado 100 mm/seg.

A2.2.3.2. EVITAR COLISIÓN

- **Prioridad:** Prioridad del comportamiento frente a los demás. A menor número, mayor prioridad. Valor predeterminado 0.
- **Timeout:** Tiempo expresado en milisegundos que permanecerá activo el comportamiento. Valor predeterminado 0, que provoca un funcionamiento indefinido

- **Sensibilidad Frontal:** Sensibilidad del comportamiento ante la presencia de obstáculos frontales. Puede tomar valores entre 0,5 (no sensible) y 3,0 (muy sensible)

- **Sensibilidad Lateral:** Sensibilidad del comportamiento ante la presencia de obstáculos laterales. Puede tomar valores entre 0,5 (no sensible) y 3,0 (muy sensible).

- **Ganancia del giro:** Establece la velocidad con la que el robot gira para alejarse de los obstáculos. Puede tomar valores entre 4,0 (giro lento) y 10,0 (giro rápido)

- **Radio de seguridad:** Define la esfera de seguridad alrededor del robot. Valor predeterminado 100 mm

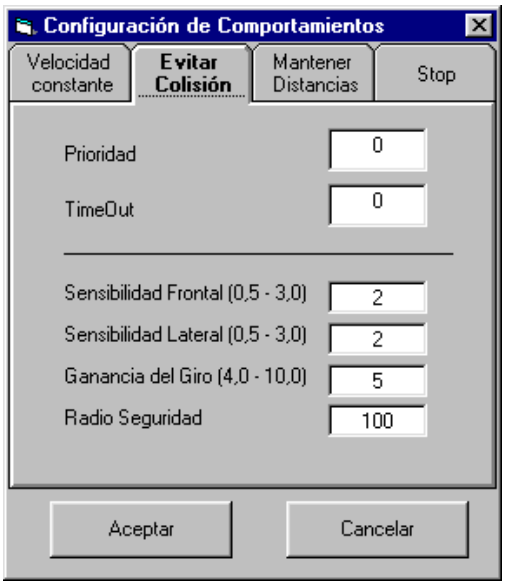


Figura 2.14: Pestaña de configuración del comportamiento Evitar Colisión

A2.2.3.3 MANTENER DISTANCIAS

Figura 2.15: Pestaña de configuración del comportamiento Mantener Distancias

- **Prioridad**: Prioridad del comportamiento frente a los demás. A menor número, mayor prioridad. Valor predeterminado 1.

- **Timeout:** Tiempo expresado en milisegundos que permanecerá activo el comportamiento. Valor predeterminado 0, cuyo significado es que permanecerá activo indefinidamente.

- **Velocidad de Precaución:** Velocidad que adquiere el robot frente a obstáculos. Valor predeterminado 30 mm/seg.

- **Sensibilidad a los obstáculos:** Sensibilidad ante la presencia de

obstáculos. Puede tomar valores entre 0,2 (poco sensible) y 2,0 (muy sensible).

A2.2.3.4 STOP

- **Prioridad**: Prioridad del comportamiento frente a los demás. A menor número, mayor prioridad. Valor predeterminado 0.

- **Timeout:** Tiempo expresado en milisegundos que permanecerá activo el comportamiento. Valor predeterminado 0, que provoca un funcionamiento indefinido

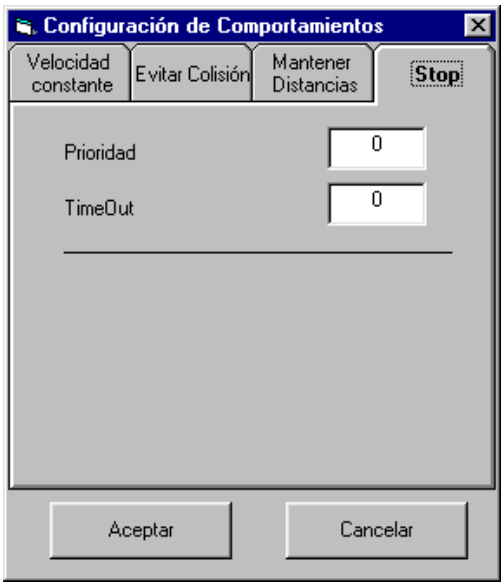


Figura 2.16: Pestaña de configuración del comportamiento Stop

A2.3. USO DEL NAVEGADOR

A2.3.1. CONEXIÓN CON EL ROBOT

Antes de realizar la conexión con el robot se ha de tener en cuenta si existe un plano predefinido de la zona en que se realizarán las evoluciones del mismo. Si es así, el primer paso a realizar es cargar el mapa en el navegador. Con este objetivo puede accederse a la ventana de carga de mapas mediante la opción *Cargar Mapa*, dentro del Menú *Ficheros*, o puede pulsarse la combinación de teclas Ctrl+M para acceder directamente.

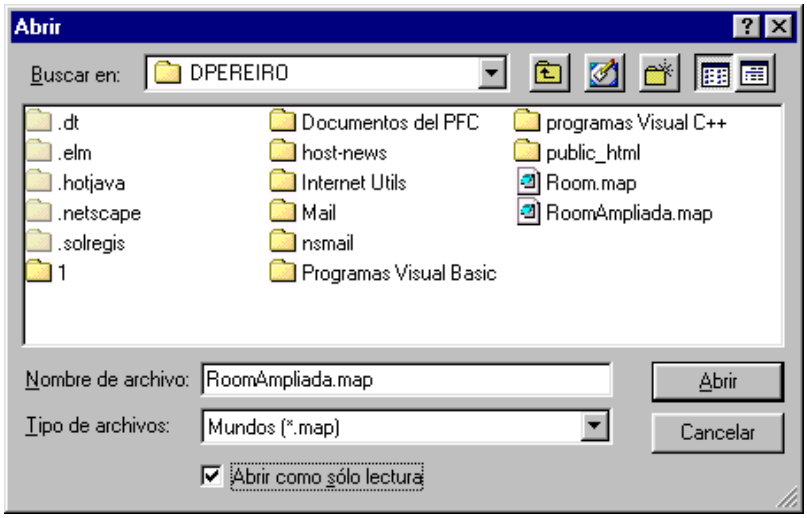


Figura 2.17: Ventana de carga de mapas predefinidos

Una vez seleccionado el mapa que representa la zona de trabajo, puede procederse a la conexión con el robot, tanto a la simulación por software como al robot real (dependiendo de la configuración, ver apartado 2.2.1.1. de este mismo Anexo), haciendo "clic" con el botón izquierdo del ratón en el icono de conexión de la botonera (ver *Figura 2.18*). Se debe tener en cuenta que este botón implica una conexión con todos los dispositivos que así estén configurados, es decir, si la opción de conectar con el Pan&Tilt está activada, el Navegador intentará conectar con él, además de con el robot.

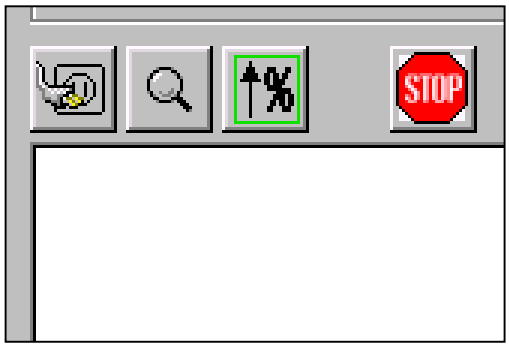


Figura 2.18: Imagen ampliada de la botonera del Navegador. El botón de conexión es el situado más a la izquierda, simbolizado por el icono de un enchufe.

Otra opción es acceder al menú *Conexión* (ver *Figura 2.19*), donde existe la posibilidad de conectarse al robot o al Pan&Tilt independientemente uno del otro.

Figura 2.19: Opciones del menú Conexión

En caso de no disponer de un mapa predefinido de la zona, no es necesario realizar ningún paso previo a la conexión, ya que el propio Navegador se encarga de generar un mapa en blanco donde seguir las evoluciones del robot.

Cualquier operación con mapas requiere la desconexión previa del robot, ya que, de otra manera, es imposible cargar un nuevo mapa o simplemente cerrar el que hay.

A2.3.2. CONTROLES DE NAVEGACIÓN

A2.3.2.1 TECLAS DE CONTROL

Las teclas que controlan el movimiento del robot en el Navegador son:

⇨	Girar robot o Pan en sentido horario
⇦	Girar robot o Pan en sentido anti horario
⇧	Avanzar robot u orientar Tilt hacia abajo
⇩	Retroceder robot u orientar Tilt hacia arriba
SPACE	Alterna el control entre el robot y el Pan&Tilt

A2.3.2.2 OPERACIONES CON EL RATÓN

El Ratón es la herramienta con la que se realizan la mayoría de operaciones del Navegador, por tanto es importante determinar cuales son las funciones del cada uno de sus botones (ver *Figura 2.20*, página siguiente), y la manera de realizarlas.

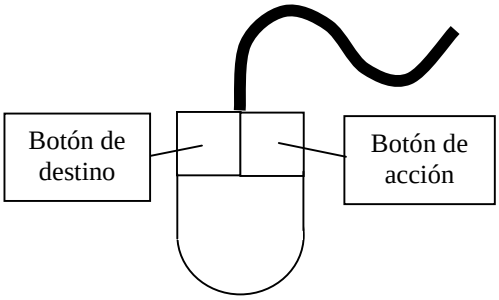


Figura 2.20: Funciones de los botones del ratón

El botón izquierdo del ratón es el encargado de las operaciones relacionadas con el proceso de generación de trayectorias. Así, este botón en conjunción con la tecla CTRL., permite delimitar la ventana de resolución, o determinar el punto de destino en el proceso de generación de trayectorias (ver apartado 3.4 en este mismo anexo).

El botón derecho del ratón, con ayuda de la tecla CTRL., realiza las operaciones morfológicas de giro y traslación del mapa.

A continuación se hace una breve descripción de las operaciones que puede realizar el usuario con el ratón:

- **Escalado:** haciendo "click" con el ratón en el icono  , aparece una barra de deslizamiento junto al mapa de navegación. Esta barra reescala dinámicamente el tamaño del mapa, que queda fijado cuando se hace "click" con el ratón en el botón "ACEPTAR"

- **Traslación:** manteniendo pulsado el botón de acción del ratón sobre el mapa, este se fija en el cursor, el cual se transforma en una mano para la operación, y es posible desplazar el mapa arrastrando el ratón. La traslación finaliza en el momento en que se deja de pulsar el botón de acción.

- **Rotación:** manteniendo pulsados el botón de acción del ratón en conjunción con la tecla CTRL del teclado, es posible hacer girar el mapa arrastrando el ratón. Si alejamos el cursor, el cual se transforma en dos flechas para la operación, del centro del mapa, este girará en sentido horario; si por el contrario, acercamos el ratón al centro del plano, este girará en sentido antihorario. La operación finaliza en el momento en que se deja de pulsar el botón derecho del ratón.

- **Determinación de un ventana de resolución:** manteniendo pulsado el botón de destino del ratón junto a la tecla CTRL del teclado, se dibuja una ventana de resolución (ver apartado 3.4 en este mismo anexo). Arrastrando el ratón sin dejar de pulsar las teclas determina el tamaño de la misma, que queda fijado en el momento en que se deja de pulsar el botón de destino del ratón. Si se realiza de nuevo la misma operación, se dibujará otra ventana de resolución que sustituye a la anterior, puesto que no es posible definir más de una al mismo tiempo. Una vez determinada, puede habilitarse y deshabilitarse su uso mediante el icono de la botonera .

Es importante notar que, la traslación y rotación del mapa, no están disponibles en caso de tener configurada una representación del mismo centrada en el robot.

A2.4. USO DE COMPORTAMIENTOS

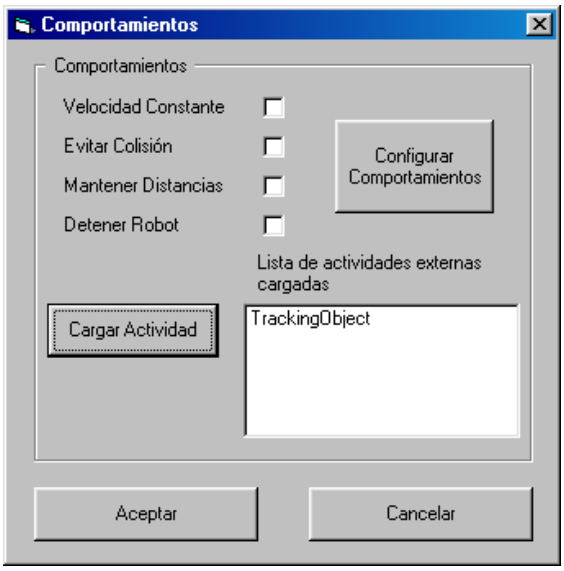


Figura 2.21: Ventana de control de comportamientos

Para activar y desactivar los comportamientos predefinidos solo es necesario acceder al menú *Comportamientos* de la barra de menús. Procediendo a la validación de las casillas de verificación adyacentes a los comportamientos, y tras hacer "click" con el botón izquierdo del ratón en "ACEPTAR", los comportamientos se activan con los parámetros de configuración fijados por el usuario (ver apartado 2.3 de este mismo Anexo). En caso de no activar o desactivar ningún comportamiento, sino simplemente modificar parámetros de configuración, estos serán aplicados a los comportamientos activos y se memorizaran los nuevos parámetros de los no activos.

Si por cualquier motivo se cierra el menú de comportamientos haciendo "click" en "CANCELAR" o pulsando la tecla "ESCAPE", el Navegador no tendrá en cuenta las modificaciones realizadas tanto en la configuración de los comportamientos, como en la activación de los mismos.

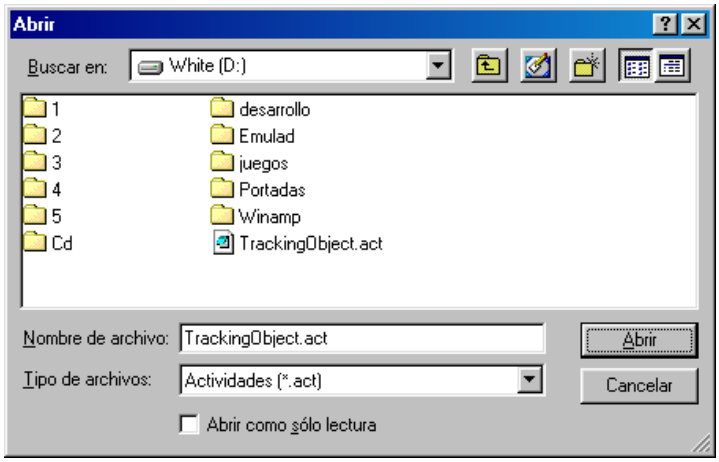


Figura 2.21: Ventana de carga de actividades externas

Para cargar actividades externas al programa de navegación, se debe pulsar el botón "Cargar Actividad". En ese momento aparece el clásico cuadro de dialogo de Windows que invita a abrir un fichero (*Figura 2.21*). Una vez seleccionado el fichero con la actividad que se desea cargar, una actividad con el nombre del fichero seleccionado aparecerá en la "lista de actividad

externas cargadas" de la ventana de control de comportamientos (ver *Figura 2.21*). A diferencia de los comportamientos predefinidos estas actividades tienen un efecto inmediato en el robot, es decir, no es necesario pulsar el botón "ACEPTAR" ya que en el mismo momento en que se procede a la apertura del fichero de la actividad, esta se pone en marcha.

Detener actividades externas es tan simple como seleccionar la actividad que queremos eliminar de la "lista de actividades cargadas", y pulsar el botón "DELETE" del teclado.

A2.5. GENERACIÓN DE TRAYECTORIAS

La generación de trayectorias es posiblemente la parte más delicada de configurar (ver apartado 2.2.2 de este mismo Anexo), aunque una vez realizado este paso previo, la tarea se reduce a un "Click" de ratón sobre el mapa.

El usuario debe hacer "click" con el botón izquierdo del ratón (botón de destino, ver apartado 2.3.2.2. en este mismo anexo) sobre una de las zonas blancas del mapa, marcando de esta manera el punto de destino del robot. A partir de este momento, el proceso es completamente automático hasta que se genera una trayectoria valida o se detiene la búsqueda por cualquiera de las restricciones fijadas por el usuario.

En caso de que el Navegador no sea capaz de conectar con éxito los puntos de partida y destino de la trayectoria, es muy conveniente definir una ventana de resolución (ver apartado 2.3.2.2. en este mismo anexo) en las zonas más conflictivas del mapa. En función de la zona que se hay delimitado con el ventana de resolución, en necesario fijar un mayor o menor porcentaje de resolución (ver apartado 2.2.2 de este mismo anexo).

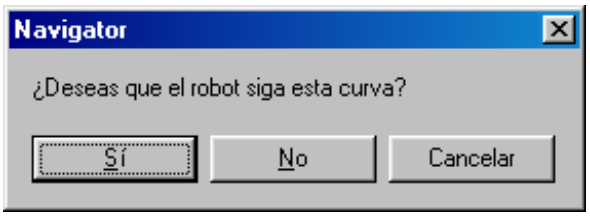


Figura 2.22

Una vez determinada la trayectoria, aparece un cuadro de dialogo como el de la *Figura 2.22*. Si la respuesta es SI, el robot comenzará a seguir la trayectoria convenida. En cambio, si la repuesta es NO, el Navegador iniciará la búsqueda de una nueva trayectoria con el mismo punto de destino y los mismos parámetros de Path Planning (si el usuario no los modifica en el menú correspondiente) que la inicial. La respuesta "CANCELAR" anula la búsqueda de trayectorias.

Una vez que se ha elegido la opción deseada, el Navegador realizará las operaciones oportunas. Si se a respondido SI al cuadro de diálogo anterior, y el robot a terminado de recorrer la trayectoria especificada, aparecerá un nuevo cuadro de diálogo que pregunta al usuario si desea guardar la imagen en disco.

Si la respuesta es afirmativa, se abrirá el clásico cuadro de diálogo de Windows que invita al usuario a elegir un nombre y un directorio para almacenar la imagen. La imagen se archiva en formato TIFF, y solo contiene una imagen del mapa, la trayectoria que debía seguir el robot, y la trayectoria seguida. No almacena ni la imagen del robot ni las lecturas de los sonares.