Sistemas Informáticos
Curso 2005/2006

Sistema Software para el Robot Guía del Museo de Informática García Santesmases.

Pablo Lanillos Pradas.
Pablo Olmos Del Amo.
Jorge Sánchez Musulín.

Dirigido por:
Dtor. José A. López Orozco.
Prof. José M. Mendías Cuadros.
Dpto. Arquitectura de Computadores y Automática.

Facultad de Informática
Universidad Complutense de Madrid

Índice

Resumen del proyecto

Resumen

El proyecto consiste en el desarrollo de un sistema para la resolución del problema de la planificación de trayectorias en dos dimensiones, para la navegación de robots móviles en superficies planas. Este sistema será utilizado por el robot guía del museo García-Santesmases de la facultad de Informática de la Universidad Complutense de Madrid. El robot tendrá la capacidad de reproducir sonidos con el objetivo de explicar a los visitantes las distintas piezas del museo.

Asimismo se desarrollará una aplicación remota mediante tecnología inalámbrica, que se conectará al robot para monitorizar su posición y estado dentro del museo y para poder controlarlo manualmente. Se creará un entorno de visualización en dos dimensiones, que simule el comportamiento del robot en respuesta a la planificación elaborada por nuestro sistema. El entorno estará compuesto por obstáculos tanto fijos como aleatorios.

Abstract

The project consists of the development of a system for solving the problem of two dimensional path planning for the navigation of mobile autonomous robots on flat surfaces. This system will be used by the robot guide of the Garcia-Santesmases museum in the Faculty of Computer Sciences of the Complutense University of Madrid. The robot will be able to reproduce sounds aimed at explaining the different equipment of the museum to its visitors.

Likewise we will develop a remote application using wireless technology, which will be connected to the robot in order to monitor its position and state within the museum, and to enable it to be manually controlled. A two dimensional visual enviroment will be created which will simulate the behaviour of the robot in response to the planning made by our system. The enviroment will be composed of different fixed as well as random obstacles.

Palabras clave

Robot, planificación, trayectorias, wifi, xml, sonido, inteligencia, artificial, remoto, modular.

Autorización

Autorizo a la Universidad Complutense a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a sus autores, tanto la memoria, como el código, la documentación y/o el prototipo desarrollado.

Firmado:

Pablo Lanillos Pradas.

Jorge Sanchez Musulin.

Pablo Olmos del Amo.

PARTE I

Descripción del proyecto.

Capítulo 1

Introducción

1. Introducción

Esta sección ofrece una primera toma de contacto con el proyecto, explicando de la manera más simple y concisa posible los primeros detalles generales que hemos considerado de interés para el conocimiento de "Sistema Software para el robot guía del museo de informática García-Santesmases".

1.1.Objetivos

El objetivo del proyecto global consiste en desarrollar por completo un robot guía para el museo de informática García-Santesmases de la Facultad de Informática de la Universidad Complutense de Madrid.

Se trata de un proyecto dividido en 3 partes diferenciadas, cada una de las cuales ha sido desarrollada por un equipo concreto (2 grupos de Sistemas Informáticos y un grupo de Ciencias Físicas). Estas partes han sido: Diseño y montaje de la plataforma física del robot , diseño e implementación de los componentes de bajo nivel del sistema que se enmarcan en la parte más cercana al hardware del proyecto (motores, sensores y montaje de los circuitos necesarios para el control de los mismos) y por último el diseño e implementación de los componentes de alto nivel del sistema que se enmarcan en la parte más cercana al software del proyecto y que tienen como objetivo realizar el control y guiado de la plataforma robótica. Esta tercera parte es la que hemos llevado a cabo nosotros y es sobre la cual versa la presente memoria.

El objetivo más importante de nuestro sub-proyecto es desarrollar toda la inteligencia que posee el robot, incluyendo el control y planificación de trayectorias a seguir, la gestión de los distintos modos de funcionamiento del robot (manual, autónomo, remoto)... . También debemos dotar al robot de la capacidad de reproducir sonido al llegar a determinados puntos objetivos dentro de su recorrido, donde explicará a los visitantes los datos más relevantes de las piezas presentes en el museo. Asimismo se desea obtener una aplicación de control remoto mediante tecnología inalámbrica (wifi) que servirá para monitorizar la posición y el estado del robot en todo momento y también para un control manual de su movimiento en caso necesario. Para la demostración de la funcionalidad se usará un entorno en 2-D que hará el papel de simulador del funcionamiento del sistema a alto nivel.

La idea fundamental ha sido partir desde cero y emplear una gran parte de los conocimientos adquiridos durante la carrera para desarrollar el sistema de control del robot, con unos resultados útiles, diferenciables, en la medida de lo posible, vistosos. Mediante la implementación, además hemos procurado desarrollar un sistema eficiente y sólido, a la vez que dotado de gran generalidad, para que el flujo de producción de la aplicación y posteriores modificaciones fuesen lo más cómodas y sencillas posibles.

1.2.Organización de la memoria

La memoria está organizada en dos grandes partes. La primera, en la que nos encontramos (parte I), se dedica a la explicación del proyecto como tal, con el propósito de proporcionar una visión general y directa de ”Sistema Software para el Robot Guía del Museo de Informática García Santesmases”. Aquí explicamos la arquitectura de tubería de la aplicación, y el posible uso futuro del trabajo realizado, entre otras cosas. Esta primera parte es la base para adquirir una visión general y completa del trabajo, necesaria tanto para conocerlo como para usarlo o ampliarlo.

Hemos añadido un segundo gran bloque a la memoria en el que detallamos los módulos que usamos, yendo un nivel mas abajo en el detalle de explicación de los mismos. Aquí damos ya una visión menos general, pero más profunda, de las diferentes tecnologías características que hemos usado para desarrollar las diferentes partes de la aplicación. Hemos intentado que este módulo de la memoria sirva como base para el uso metódico del proyecto, y que ofrezca al lector una primera

aproximación a la verdadera arquitectura e implementación del trabajo. Por tanto, es de interés para aquellos que vayan a usar el proyecto de una manera más avanzada que la de la simple prueba.

2. Especificación

A continuación se enunciaran las características y requisitos del proyecto, los objetivos deseados para su desarrollo y las funcionalidades que tendrá el proyecto. Estas definirán el posterior diseño, desarrollo e implementación de todo el sistema.

El proyecto ha sido pensado, lógicamente, a largo plazo por su dificultad. El desarrollo de un robot, tanto a nivel software como hardware, la construcción del mismo y la integración real, son factores que influirán de manera importante en la especificación. Hay que ser muy prudentes y no dejarse llevar por los deseos. También hay que tener en cuenta que es un proyecto donde participan más de 6 personas y por lo tanto las tareas de gestión de configuración e integración marcarán las pautas del trabajo. Por ello se han trazado unos objetivos que demostrarán la potencia del robot y asentarán una base sólida para próximas ampliaciones. Estas funcionalidades deberán quedar implementadas al finalizar el proyecto, y deberán ser constatadas con pruebas.

2.1. Control Remoto interno o por red

El robot se deberá poder controlar de manera remota, delegando todas las acciones al usuario. Tendrá que aceptar las acciones de usuario (parar, adelante, atrás, adelante-izquierda, adelante-derecha, atrás-derecha, atrás-izquierda, giro sobre si mismo izquierda y giro sobre si mismo derecha) como si fuera un vehículo control remoto Así mismo el usuario puede decidir que acciones autónomas realizará.

2.2.Monitorización

Tiene que poder ser monitorizado localmente y de manera remota por red inalámbrica. Es muy importante en un objeto móvil autónomo su posible monitorización, tanto para las pruebas, como para saber donde está.

2.3.Autonomía

Al ser un robot guía tiene que ser capaz de comportarse de manera totalmente autónoma, si no existe control del usuario. Por lo tanto dado una lista de objetivos y acciones tiene que realizarlas con autonomía y sin fallos. También deberá informar si hay algún incidente y recargarse si no tiene batería.

2.4.Pseudo-Inteligencia

Para poder tener una autonomía y ser versátil, tiene que poseer un cerebro que controle lo que se está haciendo y tome las decisiones pertinentes: cuándo ejecutar una acción, cuándo cambiar de objetivo, cuándo parar y cuándo moverse, cuándo cargarse, etc. Tanto controlado por el usuario como en autónomo tiene que ser capaz de registrar lo que está ocurriendo y obrar en consideración. Por ejemplo, podría ayudar al usuario en el control, o cargar nuevos objetivos para su posterior desplazamiento.

Por otro lado, en el caso de bloqueo en cualquiera de los movimientos, por causas de objetos en movimiento o lugares demasiado estrechos, tiene que ser capaz de poder escapar, esperar y en último caso avisar, o elegir otro objetivo.

2.5. Planificación de trayectorias seguras

Cada movimiento realizado tiene que ser seguro. Toda trayectoria seguida por el robot tiene que estar libre de obstáculos. Para ello se contará con una planificación off-line sobre el mundo real, y una planificación local, sobre el mundo cercano al robot. Se tiene que garantizar que no existen colisiones para evitar destrozos o deterioros del lugar y del robot.

2.6. Interacción sensorial

Necesitamos una interacción sensorial bidireccional, para interactuar con el mundo.

Teniendo en cuenta que deseamos trayectorias libres de obstáculos debemos poder reconocer el terreno con sensores y poder añadirlo a nuestro mundo. El robot tiene que poder esquivar todo obstáculo que repentinamente se le pone en medio, o cualquier objeto que se encuentre por el camino, que no estuviera contemplado.

Por otra parte el robot tiene que poder interactuar con las personas, indicando su estado e incluso hablando.

2.7. Interacción verbal

El robot tiene que ser capaz de reproducir archivos de audio. Por ser un robot de interacción con el público es necesario que tenga la capacidad de hablar. De hecho al ser un robot guía, tendrá que comentar cada elemento en su ruta de objetivos.

2.8.Simulación

Todas las funciones del robot se tienen que poder simular sin él físicamente. Así, se podrán probar los itinerarios, las acciones y los movimientos sin poner en peligro el robot, ni el entorno. Esta funcionalidad es muy importante para debugging y pruebas. El comportamiento en simulación tiene que ser lo más parecido posible a la realidad, llegando incluso a predecir el comportamiento real del robot. Así sí algo funciona en simulación, obligatoriamente tiene que desarrollarse sin problemas con el robot físico conectado.

2.9.Facilidad de Integración e Integración hardware total

El módulo software de alto nivel tiene que ser fácil de conectar al de bajo nivel, posteriormente debe ser totalmente conectado para su uso con el robot.

2.10. Funcionamiento Real

La aplicación software tiene que poder ser ejecutada en la pcbox del robot y todas las funcionalidades podrán ser realizadas en el robot físico.

2.11. Configuración XML

Normalmente para la adecuación real del sistema a un robot es necesaria la calibración de muchos parámetros de configuración. Por lo tanto todos los módulos utilizados tienen que poder ser modificados por xml, modificando así el comportamiento del robot sin compilación del programa.

El mapa y las rutas tienen que ser definidas en xml y poder ser tratadas por la aplicación del robot.

2.12. Portabilidad

Al ser un proyecto que será ampliado durante varios años, el sistema operativo no tiene que influir en la operatividad de la aplicación, ya que nunca se sabe sobre que plataforma se ejecutará.

La aplicación tiene que poder ejecutarse y compilarse en cualquier sistema operativo, con excepción de partes muy específicas, como puede ser la lectura del puerto serie o el envío de paquetes TCP.

2.13. Editor de mapas

Para facilitar el uso del robot existirá un editor de mapas que permitirá la creación intuitiva del entorno, inserción de obstáculos y objetivos. Así no hay que saber la representación interna de los mapas (que en nuestro caso es en xml). Esta aplicación será también totalmente portable.

2.14. Rendimiento y posibilidad de ampliación

El sistema desarrollado no tiene que consumir prácticamente tiempo de procesador cuando no se esta realizando ninguna acción y como mucho el 50% de procesador de un pentium III en máximo trabajo. Así mismo tiene que poder ser conectado en sistema distribuido o en multiprocesador.

Es un robot que pretende ser mejorado, así que tiene que ser posible la ampliación de las funcionalidades sin modificar el código base.

3. Estado del arte

3.1.Punto de partida

Se trata de un proyecto propuesto por el Departamento de Arquitectura de Computadores y Automática de la Facultad de Informática en el ámbito de las normas que regulan la asignatura de Sistemas Informáticos. Madurada la idea, el proceso comienza con las reuniones pertinentes con nuestro profesor director, que nos introduce en la materia, proporcionándonos principalmente la documentación y referencias pertinentes. Existen en la literatura una gran cantidad de trabajos relacionados, si bien dado que el objetivo principal del proyecto no es la investigación, nos centramos en las referencias básicas que aparecen en la sección bibliografía.

3.2.Conocimientos previos

En el desarrollo del proyecto ha sido necesario el uso e integración de conocimientos adquiridos previamente en asignaturas impartidas a lo largo de la carrera de Ingeniera Informática Superior. De esta forma el proyecto se ha fundamentado en conocimientos previos de asignaturas como Inteligencia Artificial, Robótica, Informática Gráfica, Ingeniería del Software, y los distintos laboratorios de Programación.

3.3.Visión general de los campos de estudio relacionados con el proyecto

Vamos a exponer brevemente una visión general de los principales campos de estudio que han constituido la base para el desarrollo de este proyecto. Se expondrá el contexto histórico y evolución de cada materia destacando los hitos más significativos así como una previsión de las futuras tendencias.

3.3.1. Robótica

3.3.1.1. Introducción

El origen de la robótica se difumina a lo largo de la historia debido a que se trata de un área donde se combinan la ingeniería industrial, electrónica, informática, biología, etc, siendo por tanto complejo determinar el momento histórico cuando nace como tal esta disciplina. Sin embargo podemos asumir que la aplicación práctica y comercial de los robots se hizo una realidad a lo largo del siglo veinte, gracias entre otros factores a la revolución electrónica de la miniaturización y el avance derivado del asentamiento de la informática.

3.3.1.2. Evolución hasta el robot móvil autónomo

A principios de los años sesenta se introducen en la industria, de modo significativo, los robots manipuladores como un elemento más del proceso productivo. Esta proliferación, motivada por la amplia gama de posibilidades que ofrecía, suscitó el interés de los investigadores para lograr manipuladores más rápidos, precisos y fáciles de programar.

La consecuencia directa de este avance originó un nuevo paso en la automatización industrial, que flexibilizó la producción con el nacimiento de la noción de célula de fabricación robotizada.

Los trabajos desarrollados por los robots manipuladores consistían frecuentemente en tareas repetitivas, como la alimentación de las distintas máquinas componentes de la célula de fabricación robotizada. Ello exigía ubicarlas en el interior de un área accesible para el manipulador, caracterizada por la máxima extensión de sus articulaciones, lo cual podría resultar imposible a medida que la célula sufría progresivas ampliaciones. Una solución a este problema se logra al desarrollar un vehículo móvil sobre raíles para proporcionar un transporte eficaz de los materiales entre las distintas zonas de la cadena de producción. De esta forma, aparecen los primeros *vehículos guiados automáticamente*(AGV's).

Una mejora con respecto a su concepción inicial estriba en la sustitución de los raíles como referencia de guiado en la navegación por cables enterrados, reduciéndose, con ello, los costes de instalación.

La posibilidad de estructurar el entorno industrial permite la navegación de vehículos con una capacidad sensorial y de razonamiento mínimas. De este modo, la tarea se estructura en una secuencia de acciones en la que a su término el vehículo supone que ha alcanzado el objetivo para el que está programado. Ante cualquier cambio inesperado en el área de trabajo que afecte el desarrollo normal de la navegación, el sistema de navegación del vehículo se encontrará imposibilitado para ejecutar acciones alternativas que le permitan reanudar su labor. Sin embargo, por sus potenciales aplicaciones fuera del ámbito industrial, donde resulta costoso o imposible estructurar el entorno, se les dotó, en la búsqueda de un vehículo de propósito general apto para desenvolverse en cualquier clase de ambiente, de un mayor grado de inteligencia y percepción.

Una definición correcta de robot móvil plantea la capacidad de movimiento sobre entornos no estructurados, de los que se posee un conocimiento incierto, mediante la interpretación de la información suministrada a través de sus sensores y del estado actual del vehículo.

El uso de robots móviles está justificado en aplicaciones en las que se realizan tareas molestas o arriesgadas para el trabajador humano. Entre ellas, el transporte de material peligroso, las excavaciones mineras, la limpieza industrial o la inspección de plantas nucleares son ejemplos donde un robot móvil puede desarrollar su labor y evita exponer, gratuitamente, la salud del trabajador. Otro grupo de aplicaciones donde este tipo de robots complementa la actuación del operador lo componen las labores de vigilancia, de inspección o asistencia a personas

incapacitadas. Asimismo en aplicaciones de teleoperación, donde existe un retraso sensible en las comunicaciones, resulta interesante el uso de vehículos con cierto grado de autonomía.

3.3.1.3. Tendencias futuras

Los trabajos más recientes hacen prever el establecimiento de tres tendencias principales en la robótica de aquí a unos años.

El primero, la consolidación de la robótica industrial, cada vez más sofisticada y eficiente. Sirva como ejemplo el proyecto que se está desarrollando en la universidad de Southern California consistente en un enorme sistema robot capaz de construir una casa en una única jornada de trabajo. De forma autónoma sería capaz de recoger el material de construcción depositado en la zona de trabajo y en un breve plazo de tiempo levantar muros y paredes siguiendo rigurosamente los planos diseñados por los arquitectos.

En segundo lugar, seguirán evolucionando los robots con capacidad para desempeñar tareas en diferentes tipos de entornos, especialmente en aquellos que constituyen un mayor peligro o incomodidad para los seres humanos. Así asistiremos al desarrollo de nuevos exploradores espaciales, aeronaves sin tripulación, robots con capacidad para combatir y asistir a tropas del ejército, etc. Sin embargo, es probable que en breve plazo este tipo de autómatas empiecen a entrar en el ámbito doméstico en forma de aspiradores autónomos, limpiadores de cristales, etc. Dentro de este grupo parece que se están estableciendo nuevas perspectivas según las cuales la solución no pasará por tener un único y sofisticado aparato, si no que tareas en principio complejas, serán realizadas por multitud de pequeños y simples robots que trabajarán en equipo. Este camino está llevando a la emulación de comportamientos biológicos de tipo enjambre o colonia como el que se da en algunas especies de insectos. Pongamos por ejemplo el trabajo de los ingenieros de la Universidad Libre de Bruselas, quienes han desarrollado un pequeño robot que simula el comportamiento de una cucaracha. En las pruebas previas, esta pequeña máquina consiguió infiltrarse en una colonia de estos insectos. En un futuro el objetivo será introducir varios ejemplares que consigan destacar como líderes en la jerarquía de la colonia para poder determinar el comportamiento de ésta, haciendo por ejemplo que migren de hábitat abandonando un lugar de uso humano.

Por último podemos suponer un amplio desarrollo de los robots en su sentido más romántico y más presente en la literatura de ciencia ficción(ref [17]). Hablamos de la consecución de robots de formas humanoides y con capacidad de interactuar de forma "inteligente" y "emotiva" con las personas en su día a día. En este sentido varios grupos de investigación están trabajando desde hace tiempo en un programa a largo plazo que concluya con la fabricación de este tipo de entidades. Así hay que destacar el robot Asimo de Honda, probablemente el autómata con la biomecánica más sofisticada desarrollada hasta la actualidad, o los proyectos de universidades como el MIT con Kismet o la de Carnegie Mellon, cuyos objetivos se centran en desarrollar interfaces sensibles a las emociones humanas para futuras entidades robóticas.

3.3.2. Inteligencia Artificial

3.3.2.1. Introducción

La finalidad de la inteligencia artificial consiste en crear teorías y modelos que muestren la organización y funcionamiento de la inteligencia. Actualmente, el mayor esfuerzo en la búsqueda de la inteligencia artificial se centra en el desarrollo de sistemas de procesamientos de datos que sean capaces de imitar a la inteligencia humana, realizando tareas que requieran aprendizaje, solución de problemas y decisiones.

3.3.2.2. Evolución

La Inteligencia Artificial "nació" en 1943 cuando Warren McCulloch y Walter Pitts propusieron un modelo de neurona del cerebro humano y animal (ref [6]). Estas neuronas nerviosas abstractas proporcionaron una representación simbólica de la actividad cerebral.

Más adelante, Norbert Wiener elaboró estas ideas junto con otras, dentro del mismo campo, que se llamó "cibernética"; de aquí nacería, sobre los años 50, la Inteligencia Artificial.

Se comenzó a considerar el pensamiento humano como una coordinación de tareas simples relacionadas entre sí mediante símbolos. Se llegaría a la realización de lo que ellos consideraban como los fundamentos de la solución inteligente de problemas, pero lo difícil estaba todavía sin empezar, unir entre sí estas actividades simples. Es en los años

50 cuando se logra realizar un sistema que tuvo cierto éxito, se llamó el Perceptrón de Rossenblatt. Éste era un sistema visual de reconocimiento de patrones en el cual se asociaron esfuerzos para que se pudieran resolver una gama amplia de problemas, pero estas energías se diluyeron enseguida (ref [6]).

Fue en los años 60 cuando Alan Newell y Herbert Simon, que trabajando sobre la demostración de teoremas y el ajedrez por ordenador logran crear un programa llamado GPS (General Problem Solver: solucionador general de problemas) (ref [6]). Éste era un sistema en el que el usuario definía un entorno en función de una serie de objetos y los operadores que se podían aplicar sobre ellos. Este programa era capaz de trabajar con las torres de Hanoi, así como con criptoaritmética y otros problemas similares, operando, claro está, con microcosmos formalizados que representaban los parámetros dentro de los cuales se podían resolver problemas. Lo que no podía hacer el GPS era resolver problemas ni del mundo real, ni médicos ni tomar decisiones importantes. El GPS manejaba reglas heurísticas (aprender a partir de sus propios descubrimientos) que la conducían hasta el destino deseado mediante el método del ensayo y el error.

En los años 70, un equipo de investigadores dirigido por Edward Feigenbaum comenzó a elaborar un proyecto para resolver problemas de la vida cotidiana o que se centrara, al menos, en problemas más concretos. Así es como nació el sistema experto. El primer sistema experto fue el denominado Dendral, un intérprete de espectrograma de masa construido en 1967, pero el más influyente resultaría ser el Mycin de 1974. El Mycin era capaz de diagnosticar trastornos en la sangre y recetar la correspondiente medicación, todo un logro en aquella época que incluso fueron utilizados en hospitales (como el Puff, variante de Mycin de uso común en el Pacific Medical Center de San Francisco, EEUU).

Ya en los años 80, se desarrollaron lenguajes especiales para utilizar con la Inteligencia Artificial, tales como el LISP o el PROLOG. Es en esta época cuando se desarrollan sistemas expertos más refinados, como por el ejemplo el EURISKO. Este programa perfecciona su propio cuerpo de reglas heurísticas automáticamente, por inducción.

3.3.2.3. Estado Actual

En la actualidad se tiende a representaciones explícitas del conocimiento específico para cada domino. Hay un auge de los sistemas expertos y de los basados en conocimiento y a su vez se ha producido un regreso de las redes neuronales y los algoritmos genéticos. Se ha realizado un gran esfuerzo y avance en el uso de sistemas inteligentes para aplicaciones reales (medicina, finanzas, ingeniería...).

Capítulo 2

Diseño y arquitectura

1. Introducción

El diseño es la parte más importante de una aplicación a gran escala y en este apartado se explicará de forma precisa para su posterior implementación. En nuestro caso necesitamos una arquitectura sólida sobre la que el diseño se sustente, y que en gran medida definirá la forma de implementación.

Una arquitectura software se selecciona y diseña en base a unos objetivos y restricciones. Los objetivos son aquellos prefijados para el sistema de información, pero no solamente los de tipo funcional, también otros objetivos como el mantenimiento, flexibilidad e interacción con otros sistemas de información. Las restricciones son aquellas limitaciones derivadas de las tecnologías disponibles para implementar sistemas de información. Unas arquitecturas son más recomendables de implementar con ciertas tecnologías mientras que otras tecnologías no son aptas para determinadas arquitecturas

Nuestra arquitectura esta basada en un modelo de tubería desarrollado por alumnos de la Facultad de Informática.

2. Arquitectura.

Teniendo en cuenta la especificación, necesitábamos una arquitectura, modular, rápida y con facilidad de intercambio de información. Así mismo cada módulo tiene que ser altamente independiente. Para ello se ha desarrollado una arquitectura basada en dos arquitecturas conocidas: tubería y sistema distribuido.

2.1.Componentes

Los distintos componentes de la arquitectura son:

- **Módulos** básicos independientes que recogen información en la entrada y la depositan en la salida.
- **Controlador**. Se encarga de cargar los módulos, conectarlos y pararlos. También controla las interrupciones, los errores y el reloj.
- **Programa principal**. Se encarga de ejecutar el controlador, inicializar el sistema y terminarlo.

2.1.1. Modulo

Cada módulo puede tener puertos de entrada y de salida configurables por el controlador. El funcionamiento interno es transparente al programador. Tanto las lecturas como las escrituras están diseñadas como modelo **productor-consumidor** y son bloqueantes. Así conseguimos no consumir tiempo de procesador si no existen datos en los puertos.

En todo modulo hay que tener en cuenta las siguientes funcionalidades:

- Iniciar: Inicializa el módulo.
- Terminar: Termina el módulo
- Leer: Lee de un puerto la información
- Escribir: Escribe un dato en el puerto de salida
- Ejecutar: Bucle donde se ejecutará las acciones como si fuera el hilo de ejecución principal.
- Lanzar: Lanza una excepción
- Interrupción: Recibe una interrupción

También se ha hecho uso del modelo de servicios, implementados como librerías estáticas, luego todo módulo tiene acceso a ellas si las importa. Se utiliza por ejemplo en el caso de búsquedas en grafos.

2.1.2. Controlador

Es un sistema multiplataforma que tiene acceso a cada uno de los módulos. Lee una configuración y en base a ella ejecuta en paralelo los diferentes módulos.

Funciona con módulos compilados desde cualquier lenguaje estándar como bibliotecas dinámicas. Esto dota a la aplicación de un marco muy amplio de uso en muchos ámbitos de desarrollo, no solamente el propio de la robótica, sino en cualquier aplicación que necesite una arquitectura modular, ya sea secuencial o paralela.

El controlador o pipeline se encarga de gestionar las interrupciones y la configuración.

2.1.2.1. Interrupciones

El sistema de interrupciones se basa en interrupciones hardware, pero llevadas al alto nivel. Todo módulo puede lanzar interrupciones, el controlador se encarga de parar la ejecución de todos los módulos que no acepten esa interrupción y llamar a los que si para que ejecuten la acción pertinente asociada a ella.

2.1.2.2. Configuración

Control de proyectos de aplicación dinámicos, mediante definición de los mismos en XML. De esta forma, diferentes archivos de configuración de proyecto pueden crear aplicaciones totalmente distintas sin tener que reimplementar nada. Así se puede variar del modelo tubería al distribuido, con un simple xml.

2.1.2.3. Ejemplos

Podemos ver en los ejemplos la versatilidad y la potencia que ofrece la configuración por XML. Los ejemplos contemplan dos aplicaciones diferentes y su definición en el XML.

Ejemplo 1:

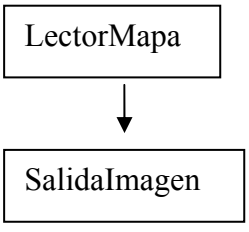
Solo existe un módulo que funciona en solitario y que no tiene ni salidas ni entradas. Este módulo es LectorMapa.

```
<pipeline>
    <modulo nombre="lectorMapa" ruta="lectorMapa" estado="activo">
    </modulo>
</pipeline>
```

Ejemplo 2:

La aplicación esta formada por dos módulos y tienen una salida conectada con la entrada del otro. Además poseen argumentos de configuración propios que definirán su comportamiento.

```
<pipeline>
  <modulo nombre="lectormapa" ruta="lectormapa" estado="activo">
      <argumento nombre="ruta_mapa_xml">xml/mapa.xml</argumento>
      <argumento
nombre="ruta_guia_xml">xml/mapaGuia.xml</argumento>
      <argumento nombre="acepta_peticiones">0</argumento>
      <argumento nombre="acepta_obstaculos">0</argumento>
      <salida puerto="puerto">2</salida>
  </modulo>
  <modulo nombre="salidaImagen" ruta="salidaImagen"
estado="activo">
      <argumento nombre="pixcm">10</argumento>
      <conexion origen="puerto"
destino="puerto"cola="2">lectormapa
      </conexion>
  </modulo>
</pipeline>
```



3. Diseño

Mostramos a continuación un diagrama UML de los elementos de diseño más importantes que guían la arquitectura.

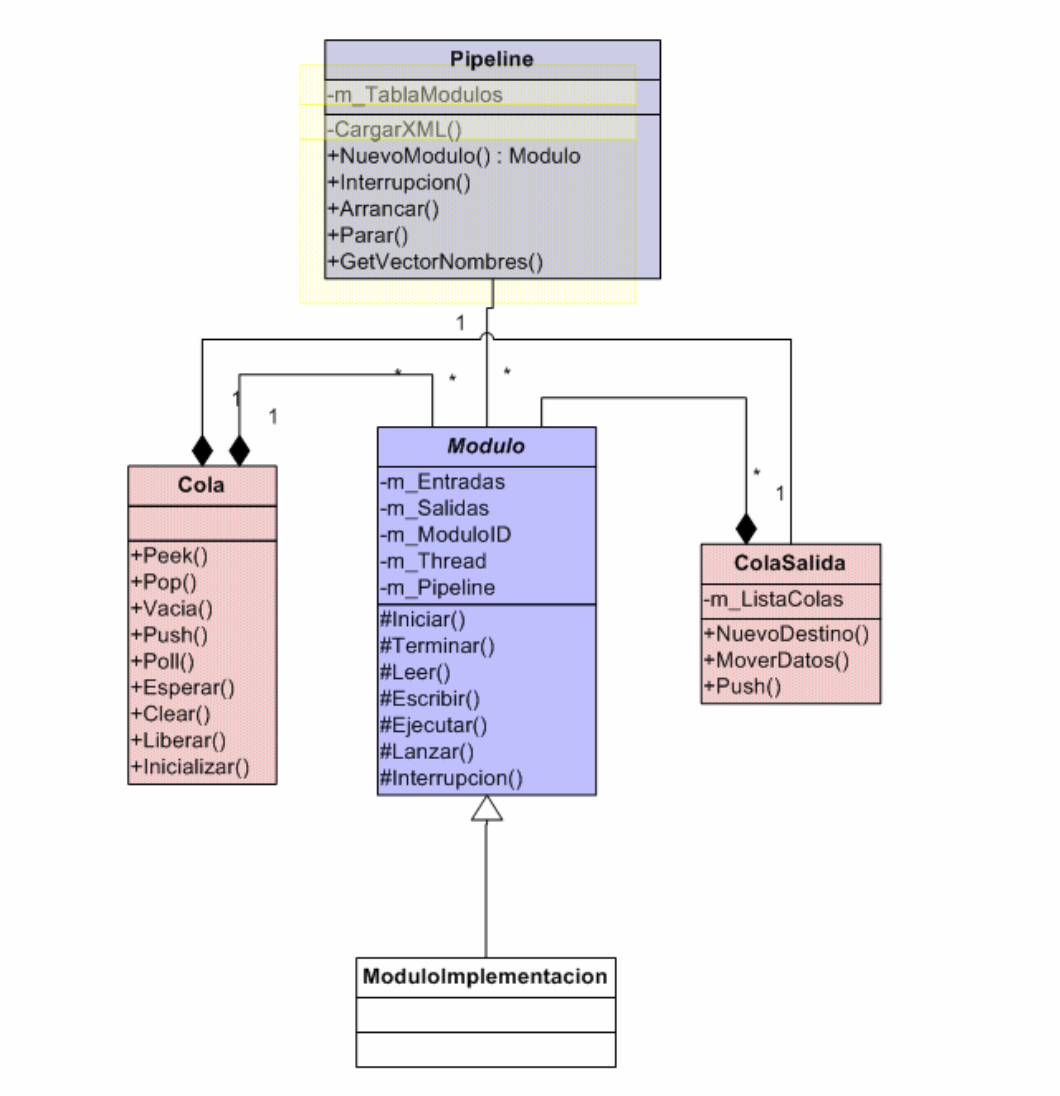


Fig 1. Esquema UML de diseño de la Arquitectura: Herencia de módulo, colas…

El Controlador (Pipeline) tendrá acceso total a todos los módulos, es decir será una friend class (lenguaje C) y los módulos tendrán la posibilidad de acceder al controlador, como si fuera su padre. Solo hace falta heredar de Modulo para crear un nuevo módulo e implementar las funciones necesarias (abstractas).

Cada Módulo puede tener 1..N colas de entrada y 1..N colas de salida. Las colas de entrada se implementarán como colas simples bloqueantes. Las de salida como cola de colas simples, es decir como entrada una cola simple y como salida una lista de colas.

Mostramos en la siguiente figura un esquema de diseño al más alto nivel de abstracción.

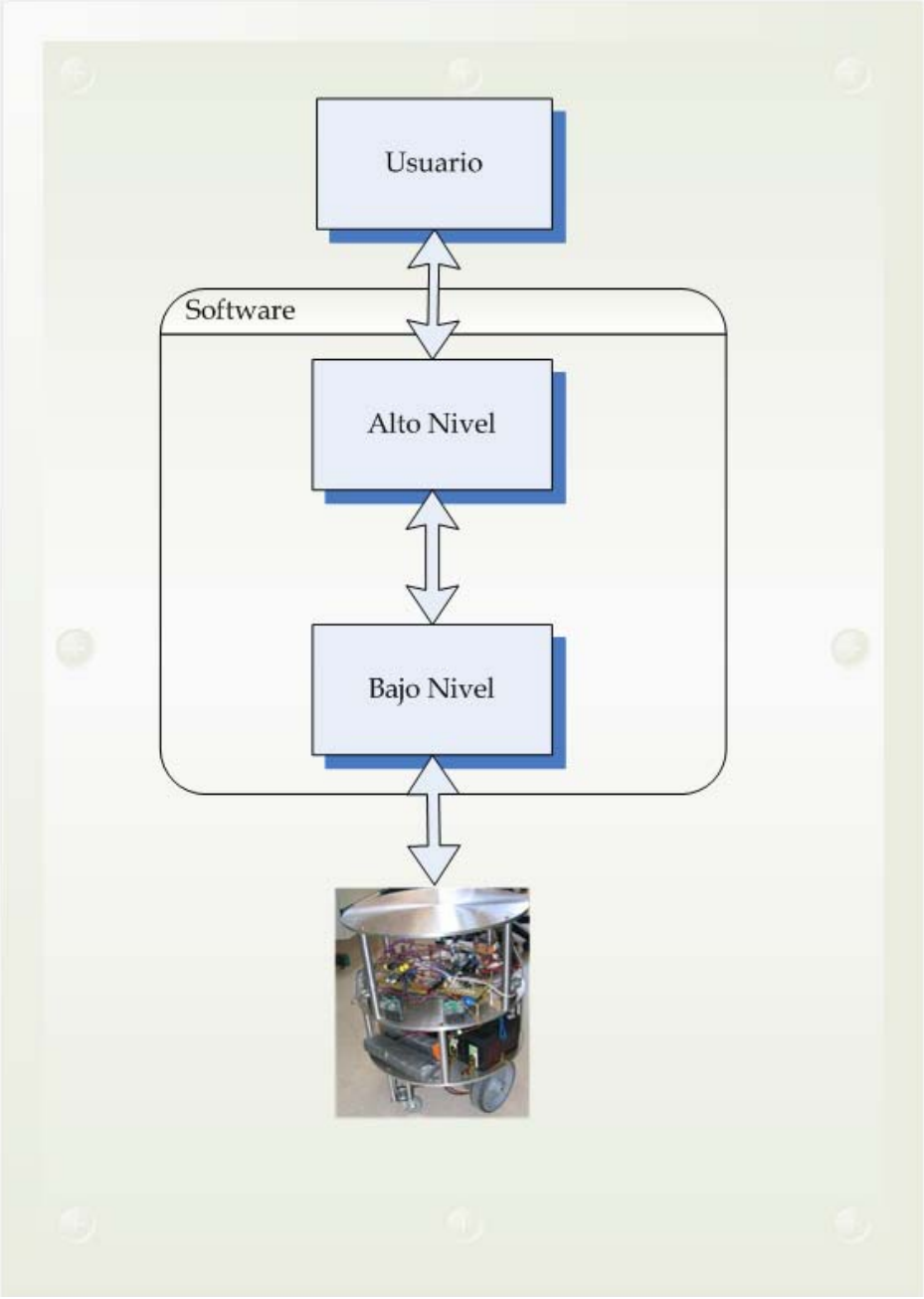


Fig 2. Pequeño diagrama general de diseño

Para el diseño final construimos un boceto que se acerca a la solución y nos da pautas de cómo afrontar la situación:

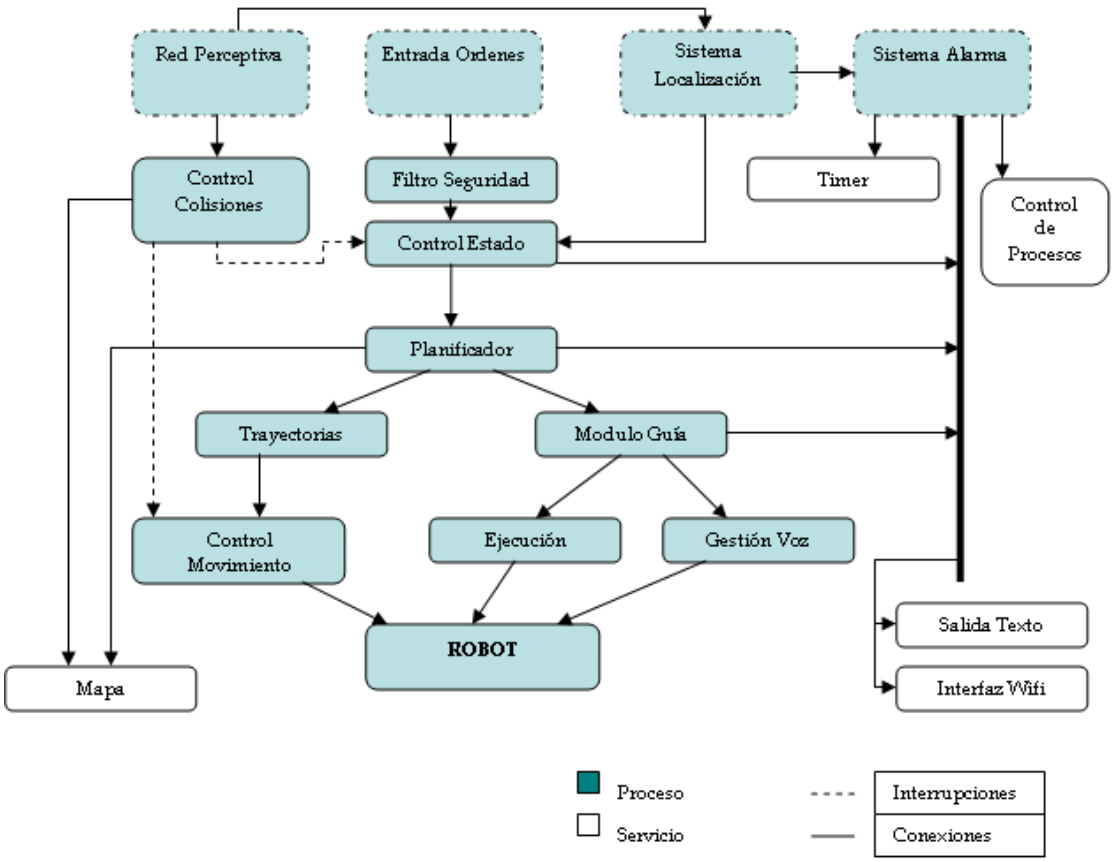


Fig 3. Primera aproximación al diseño final

Este diseño tiene un fallo grave. No se define bien la separación entre el software de bajo nivel y el de alto nivel. Para ello definimos cada uno.

El diseño de la parte de alto nivel comprende desde la lectura del entorno e interacción con el usuario hasta la determinación de las acciones, la definición de las mismas, la generación de trayectoria en puntos del espacio cartesiano y la determinación de la velocidad del robot.

El trabajo de bajo nivel comprende todas las etapas que están relacionadas directamente con el robot físico, como la transformación de movimientos de alto nivel (lista de puntos separados x cm) a velocidades de las ruedas y a su vez a órdenes entendibles por el hardware.

En esta parte del diseño nos centraremos en lo que hemos llamado alto-nivel, y comentaremos por encima el resto.

3.1.Diseño Alto-Nivel No Remoto

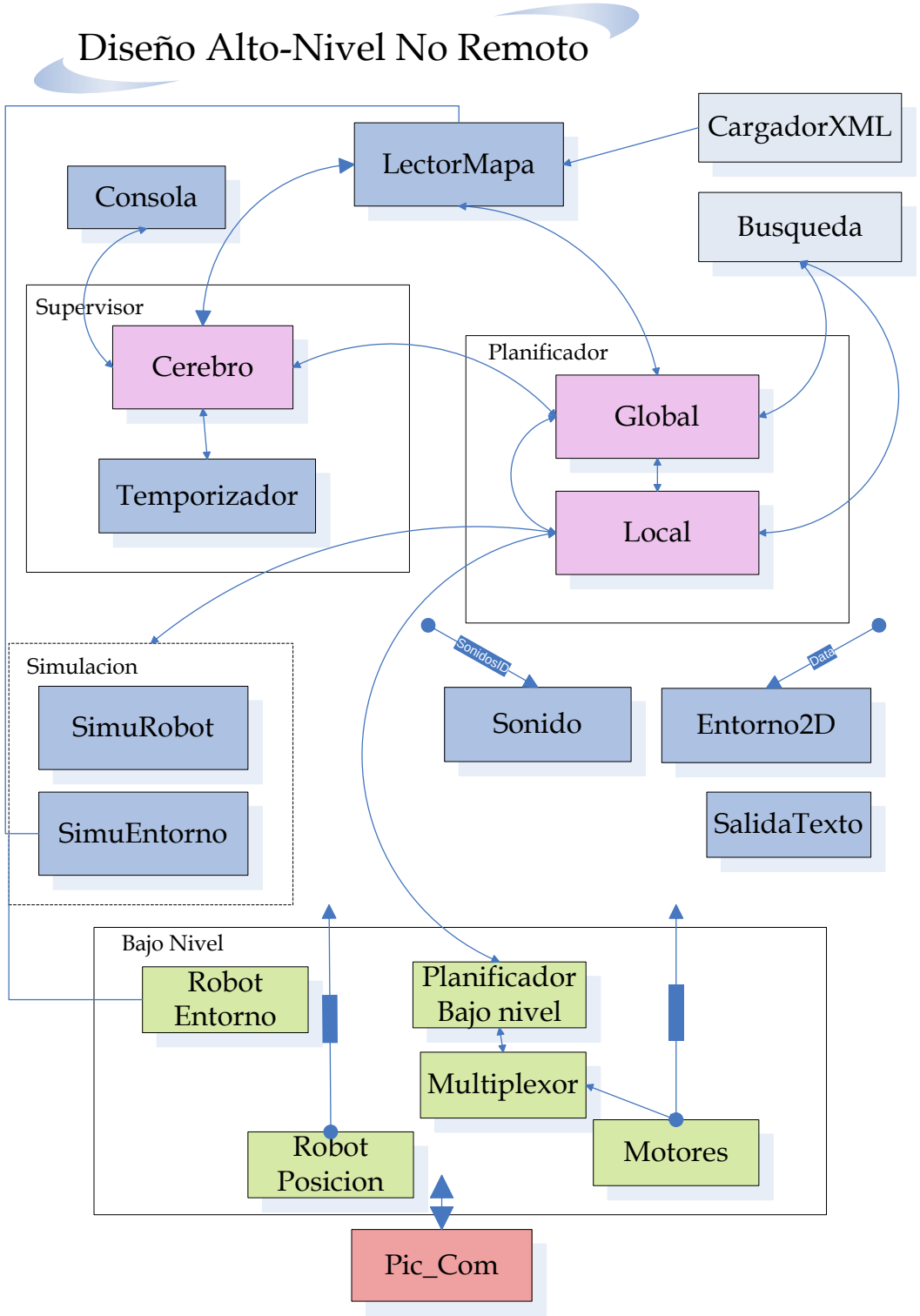


Fig 4. Diseño Alto-Nivel No Remoto

La aplicación diseñada sin salida remota. Es decir, cuando el control del robot se hace de manera local.

3.2.Diseño Alto-Nivel Remoto

El control del robot se hace de manera remota por medio de una conexión wifi.

El robot posee un servidor que recibe y envía toda la información necesaria.

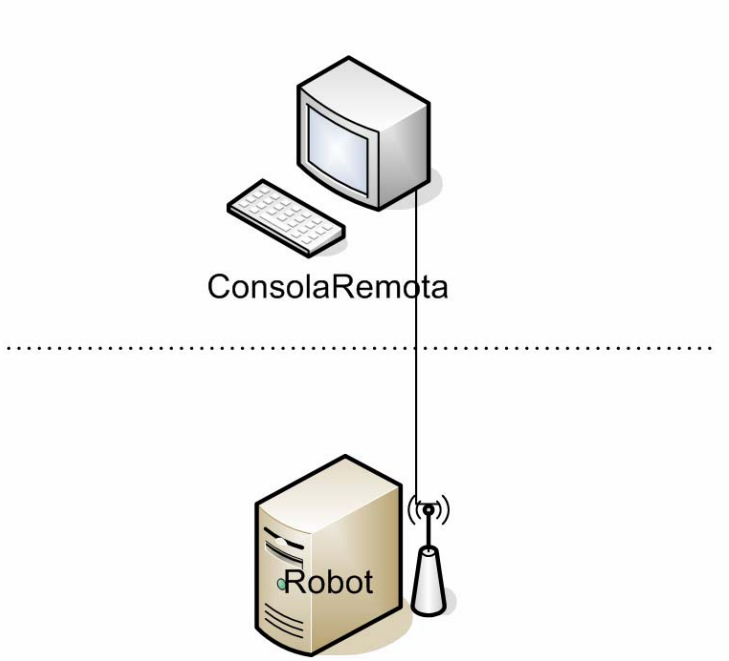


Fig 5. Conexión ConsolaRemota-Robot

La conexión por Wireless con el robot nos permite monitorizar su posición y controlarlo desde otro lugar sin cables.

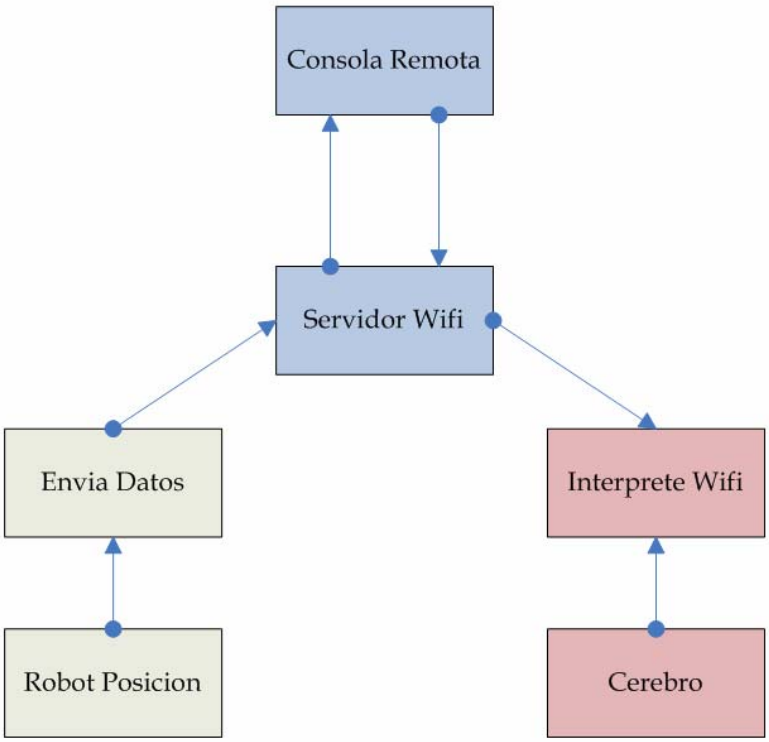


Fig 6. El diseño interno del sistema de conexión por red

3.3.Diseño por Bloques

El diseño se divide en 5 grandes bloques.

3.3.1. Utilidades

Son librerías estáticas que funcionan como servicios para los otros bloques.

3.3.1.1. Librería de búsqueda

La librería más relevante que usamos es una librería que contiene las implementaciones necesarias para las búsquedas en grafos que necesita realizar el bloque planificador. Asimismo contiene las representaciones adecuadas de la estructura de grafos que utiliza.

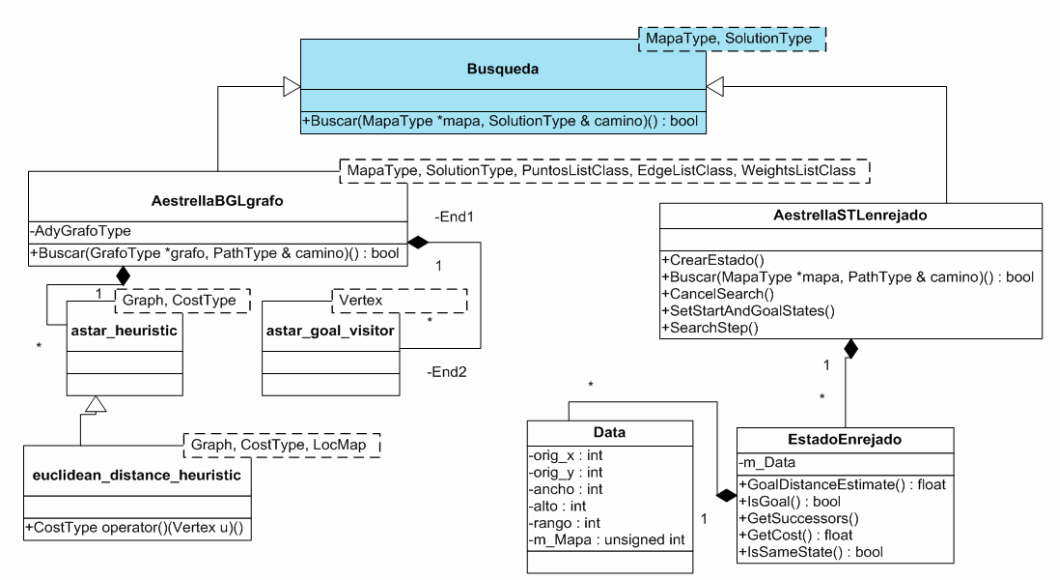


Fig 7. Uso de la librería de busqueda

3.3.1.2. Cargador XML

Lee la información de un fichero XML y la transforma a datos reconocibles por los módulos.

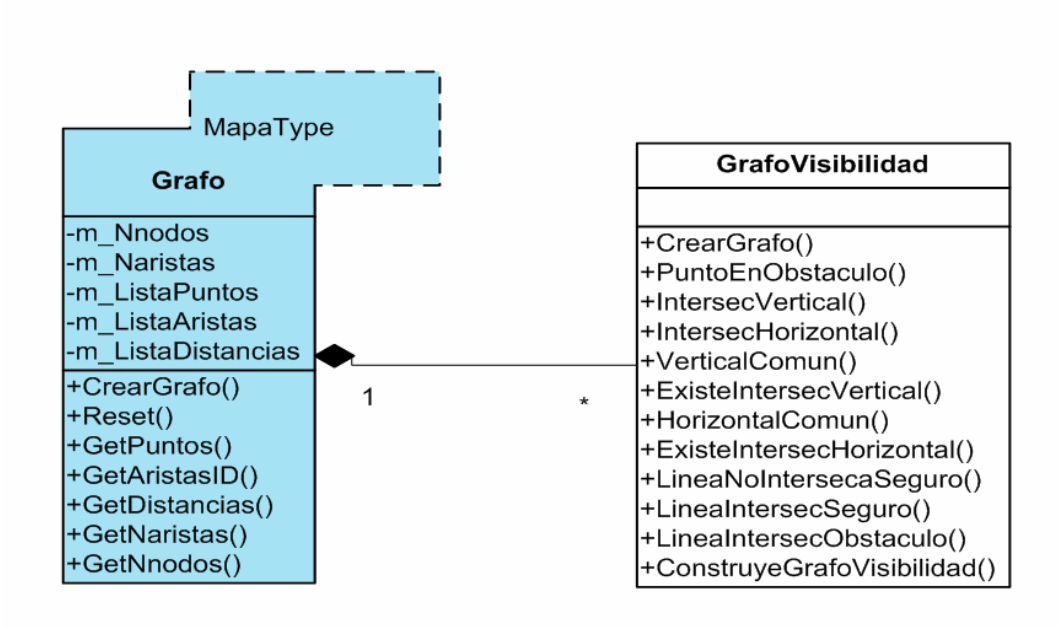


Fig 8. Diagrama UML Búsqueda y Grafo

3.3.2. Entrada/Salida, Interfaz con el mundo

Comprende todos los módulos que a partir de elementos externos se convierten en órdenes y mapas entendibles por el robot, y módulos que a partir de elementos internos se convierten en datos entendibles por las personas. Este apartado lo integran los siguientes módulos:

3.3.2.1. Módulos de entrada

- Consola: Consola en modo texto, para interactuar con el robot de manera local.
- Consola Ventana: Consola en modo de interfaz amigable, para interactuar con el robot.
- Lector Mapa: Modulo que se encarga de guardar toda la información del mapa y el entorno. Centraliza todo el control sobre el entorno virtual donde se mueve el robot.

- Interprete Wifi: Traductor de la información que le llega del servidor wifi, que se encarga de enviársela a los diferentes módulos.

3.3.2.2. Módulos de salida:

- Salida Texto: Muestra la información del robot en modo texto en una consola
- Salida Ventana: Muestra la información del robot en modo texto pero en una interfaz amigable.
- Enviar Datos: Traductor de los datos del robot a sentencias entendibles por el servidor wifi, para su posterior envío.
- Entorno 2D (Salida Imagen): Ventana donde se muestra el mundo donde se mueve el robot y la simulación de su movimiento de manera gráfica.

3.3.2.3. Módulos de Entrada/Salida:

- Servidor Wifi: Modulo encargado de la conexión en red para un usuario remoto.

3.3.3. Núcleo del Sistema

Los módulos que integran este bloque son aquellos que permiten el cálculo de trayectorias, la pseudo-inteligencia, monitorización, etc, es decir los módulos principales del robot.

3.3.3.1. Supervisor

En la especificación hablábamos de la pseudo-inteligencia. Pues en este bloque se integra la organización temporal y el cerebro de la aplicación. Todas las entradas de la interfaz con el mundo envían información a este bloque. Está compuesto por los siguientes módulos.

- Cerebro: Se encarga de gestionar las acciones y el comportamiento del robot. Hace de puente con el usuario.
- Temporizador: El encargado de controlar el tiempo.

3.3.3.2. Planificador

Es el bloque encargado de generar trayectorias libres de obstáculos a partir de peticiones del supervisor. Los módulos que lo integran son:

- Planificador Global: Planifica de manera global las trayectorias sobre un mapa dado.
- Planificador Local: Planifica en el entorno comprendido por los sensores, para poder esquivar obstáculos

3.3.3.3. Mapa

Módulo encargado de la centralización y control del entorno donde se desplaza el robot, es decir el mapa y sus obstáculos. En él se encontrarán todas las funciones de lectura de mapas y funciones de modificación. Así mismo contendrá todas las representaciones necesarias para los otros módulos. Es un módulo que puede servir tanto de interfaz con el mundo como núcleo del sistema.

3.3.3.4. Simulación

Bloque encargado de simular el robot de bajo nivel, cuando éste no existe.

3.3.4. Unión Software-Hardware

Es el módulo encargado de transformar los datos para que el hardware lo entienda y recibir la información del hardware para procesarla. Utilizamos una comunicación serie emulada con un microcontrolador pic mediante conexión usb.

3.3.4.1. Pic Com

Comunicación de la parte software por puerto COM con el pic.

3.3.5. Aplicaciones Externas

Son aplicaciones que no están manejados por el controlador, y se pueden ejecutar en otros ordenadores. Son Consola Remota y Editor XML.

3.3.5.1. Consola Remota

Consola diseñada para interactuar con el robot desde otro ordenador.

3.3.5.2. Editor XML

Aplicación para crear y modificar mapas.

Capítulo 3

Resultados del Proyecto

1. Pruebas

1.1.Pruebas de la arquitectura

La primera parte del proyecto residía en la creación de una arquitectura sólida que nos diera robustez al diseño de la aplicación. Por lo tanto incluiremos los diferentes test realizados.

1.1.1. Aplicación Productor-Consumidor

La primera prueba residía en realizar una aplicación productor consumidor que funcionara sobre la arquitectura. Así, un módulo enviaría información a un puerto y el otro lo recogería y lo mostraría por pantalla.

Para ello usamos una entrada y salida simple, es decir sin colas. Tanto la carga desde el XML como la ejecución de los mismos funcionaba correctamente pero el consumo de CPU era muy elevado, puesto que los procesos intentaban leer o escribir todo el tiempo.

```
Robot iniciado...
Cargando Pipeline...
Pipeline:Cargando Modulo: productor
Pipeline:Cargando Modulo: consumidor
Pipeline:CrearConexiones: origen:puerto_entrada
destino:puerto_palabra
Mostrando Modulos...
consumidor
productor
Pipeline Corriendo...
Productor arrancado
Consumidor arrancado
```

Así que implementamos una estructura de colas tanto de entrada, como de salida que fueran bloqueantes. Esto se consiguió usando mutex.

Finalmente la conexión entre Módulos se implementó para conexiones N a N. Para ello un módulo tiene tantos puertos como conexiones entrantes y salientes, y asociadas a ellos hay colas de entrada y colas de salida respectivamente. Para dotar de más potencia al sistema, diferentes módulos se pueden conectar al mismo puerto de otro.

Dado que el texto producido por el productor es: *Independiente (Productor)*, el resultado de la aplicación es el siguiente:

```
Leer del puerto
Dato recogido del Puerto
Vuelta en Independiente (Consumidor): Independiente (Productor)
Leer del puerto
Dato recogido del Puerto
Vuelta en Independiente (Consumidor): Independiente (Productor)
Leer del puerto
Dato recogido del Puerto
Vuelta en Independiente (Consumidor): Independiente (Productor)
```

Para hacer la tarea más visual realizamos un programa principal con interfaz amigable para poder parar, arrancar y ver como se desarrollaban las gestiones de la arquitectura.

Fig 9. Programa principal

1.1.2. Interrupciones

Era importante demostrar que las interrupciones funcionaban correctamente y que el controlador y los módulos reaccionaban rápido. Por eso, redefinimos la configuración por xml añadiendo una sentencia interrupción para saber qué módulos aceptan qué interrupciones y desde qué módulos.

Definiendo <interrupcion>consumidor</interrupcion>

```
> INICIANDO INTERRUPCION lanzada por: consumidor
 -> productor ACEPTA lanzada por: consumidor
productor Tipo[Letras] RECIBE INTERRUPCION: 0, 0x357604e.
productor Tipo[Letras] FIN PROCESAR: 0, 0x357604e.
 > FIN INTERRUPCION PIPELINE lanzada por: consumidor
```

Finalmente instanciando varios módulos productores y consumidores unos dependientes y otros no, comprobamos como solo se paraban los módulos que no aceptaban esa interrupción, hasta que esta era procesada por los que si la aceptaban.

1.1.3. Gestión de errores

Nos dimos cuenta que cuando esta mal definido un puerto, es decir que posee un nombre erróneo o que simplemente no se ha definido en el xml, se producían violaciones de memoria. Por eso decidimos comprobar de manera transparente al programador si ese puerto era válido, y en caso contrario avisar al usuario.

```
Error Modulo: Acceso Ilegal a Puerto Entrada:puerto_entrada
```

1.1.4. Limpieza de puertos y finalización

El último retoque obligado al intentar parar la aplicación era conseguir que, aunque los módulos tuvieran las lecturas y escrituras bloqueantes, el programa saliera sin que hubiera datos en el puerto. Además era necesaria la posibilidad para el programador de poder borrar todos los datos que le hubieran llegado al puerto, por posibles bloqueos o reinicializaciones.

1.1.5. Eficiencia

El pipeline no es un ejemplo de velocidad de proceso. Las pruebas que hemos realizado nos han permitido, en un ordenador con un microprocesador Intel Pentium IV con una frecuencia de reloj de 3,0 GHz, alcanzar velocidades de ciclo cercano al tiempo de ciclo del procesador. Aunque poniendo un límite de 10 milisegundos si no se realizan cálculos extras de alto coste como podrían ser módulos gráficos o operaciones matemáticas costosas, es una arquitectura ampliamente preparada para tiempo real, y por lo tanto una base sólida para la realización de un robot real. Y lo más importante es que cuando no existen datos en los puertos el consumo de CPU es mínimo, contando solo la carga de la Arquitectura y no las posibles operaciones de los módulos.

1.2.Comprobación de los módulos

Solo se analizarán los módulos que sean por su complejidad ó tiempo de cálculo interesantes para ver su funcionamiento sobre la arquitectura. Aun así hay que decir que si el módulo por separado tiene un coste c, dentro de la arquitectura tendrá un coste c + una constante mínima.

De todas maneras en la parte II de esta memoria se podrán observar ejemplos concisos del funcionamiento de la totalidad de los módulos de la aplicación.

1.2.1. Búsqueda

Uno de los problemas de la robótica autónoma es el cálculo de trayectorias en tiempo real, para dotar al robot de una simulada inteligencia. Así se decidió que las búsquedas no serían módulos específicos, sino librerías estáticas las cuales se podrían usar simplemente importando la librería y las cabeceras. Con esto la arquitectura no influye sobre su rendimiento.

1.2.1.1. Búsqueda sobre grafos:

Desde el principio pretendimos buscar una buena librería con una implementación del A* muy rápida. La encontramos dentro de una librería gráfica. Aún siendo una búsqueda off-line, es decir, que no tiene que ser en tiempo real, las medidas realizadas, demuestran que podría ser utilizada para tiempo real cuando el grafo de búsqueda tenga menos de 1000 nodos, siendo inapreciable para el ser humano. El problema reside en la construcción del grafo, que es el proceso de implementación en el que se consume más tiempo de procesador. Mostramos un ejemplo de ejecución de búsqueda sobre grafo.

```
Aristas: 152
Grafo Nodos: 26
--------- A* ---------
num_edges: 152
num_nodos: 26
Start vertex: 0
Goal vertex: 1
Shortest path from 0 to 1: 0 -> 3 -> 12 -> 1
Total travel distance: 50.9589
```

1.2.1.2. Búsqueda sobre Celdas

El diseño ha sido realizado a medida para tiempo real, pero siempre si hablamos de espacios de estados pequeños como los que podría ser el alcance de unos sensores de ultrasonidos en centímetros: 300x300.

1.2.2. Cerebro

El cerebro fue probado, simulando las entradas y viendo como reaccionaba a los estímulos, aceptaba las transiciones y cambiaba de estado.

También se comprobó como funcionaba cuando el usuario tomaba control sobre el sistema y si el cerebro dejaba de hacer toda acción pertinente.

1.2.3. Temporizador

En principio era un módulo importantísimo para la aplicación pero luego se vio que realmente era secundario e incluso omitible. Aún así le proporciona una base potente para posibles ampliaciones. El test se realizó probando varias peticiones con diferente número de ejecuciones, desde una vez, hasta ilimitado. Y se pudo comprobar que la gestión era correcta y que el desfase temporal que podía haber entre la realidad era de 10 microsegundos, en circunstancias normales. La única limitación es que funciona con señales Glib(ref [18]) y sin la libreria y sus inicializaciones correctas no funciona.

1.2.4. Planificador Global

Para el planificador, primero se ejecutaron pruebas estáticas de la búsqueda con un grafo con propiedades controladas, luego se usó un grafo generado dinámicamente, teniendo que acoplar los tipos de entrada del algoritmo A* con los generados por nosotros, es decir crear una lista de adyacencia. La búsqueda es óptima con la heurística de distancia euclídea con respecto a las distancias entre puntos del plano.

Una vez hechos esos test se probó el módulo sin conexiones a puertos, simplemente simulando una entrada y mostrando la salida. Finalmente se unió al cerebro para que la entrada fuera dinámica. La integración no tuvo problema alguno.

Así llegamos a mostrarlo en una imagen para ver intuitivamente el funcionamiento del planificador. En la imagen la línea verde es la generada por el planificador global.

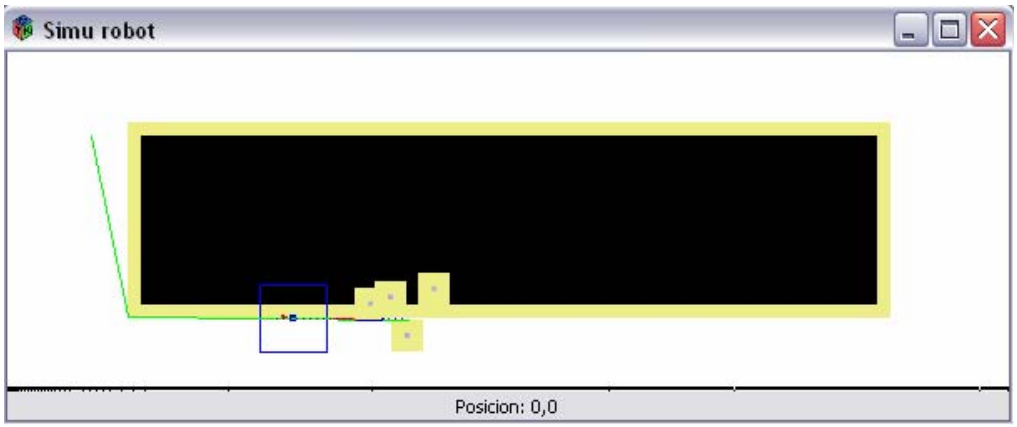


Fig 10. Simulación del robot

1.2.5. Planificador Local

El planificador Local dio muchos problemas y aún ahora se le podrían hacer múltiples mejoras. El problema residía en manejar el comportamiento del robot de manera inteligente, pero rápida. Además había que encontrar una representación para el mundo que fuera compatible con los datos recibidos de los sensores.

Uno de los inconvenientes más grandes era el solapado, los bloqueos y la simulación del entorno:

Solapado. Fue solucionado por medio de parámetros de ajuste.

Bloqueos. De una manera intuitiva se optó por la reducción de la distancia de seguridad, la replanificación y en último caso avisar al supervisor.

Simulación del entorno. Realizar un buen modelado de sensores es una tarea muy complicada y se salía de nuestro proyecto. Por eso optamos por la inclusión de un módulo que insertaba objetos aleatoriamente. Sin embargo esto nos provocó grandes dolores de cabeza, ya que aparecían en medio del robot o rodeaban al robot. Para solucionarlo, aparte del bloqueo, optamos por usar un factor de olvido, que explicaremos más adelante en la parte II de este documento, en el módulo LectorMapa.

Gracias a múltiples parámetros de configuración, se puede ajustar el planificador para que funcione en cualquier entorno.

La velocidad de proceso cuando no hay objetos es casi instantánea, ya que solo se divide la trayectoria en puntos consigna, separados a x centímetros, por medio de operaciones matemáticas sencillas. Y cuando hay obstáculos no nos importa mucho la velocidad, ya que tenemos que ralentizar el robot o incluso esperar.

La fusión del planificador global y local es trivial, ya que son dos módulos totalmente independientes, con excepción de la replanificación. Aún así se ha hecho hincapié en que los dos módulos puedan funcionar sin el otro. Es decir, el robot podría con unos buenos sensores desenvolverse con el planificador local, y si el entorno fuera absolutamente fijo, el global sería suficiente para el robot.

En la Fig 10 se puede observar como la línea roja es la planificación local y la línea azul la parte de ella que el robot sigue.

1.2.6. LectorMapa

Para la lectura del mapa se pensó en la representación de éste como una imagen BMP (bitmap), pero luego las modificaciones y los algoritmos de lectura no eran buenos. Para las modificaciones necesitábamos, por ejemplo, cambiar en un solo un objeto unas determinadas coordenadas, y editar un BMP para solo modificar unas coordenadas era una locura.

Por eso decidimos que el mapa fuera en formato XML, así continuamos con el estándar de la aplicación y permitimos una portabilidad y una versatilidad acorde con la aplicación.

Fig 11. Mapa en .bmp

Versión XML de otro mapa más sencillo:

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapa (View Source for full doctype...)>
      <mapa alto="2000" ancho="6000" id="planta3" pared="0"
pxcm="10" ruta="">
<objetivos />
<obstaculos>
<obstaculo>
<punto x="800" y="500" />
<punto x="5200" y="500" />
<punto x="5200" y="1500" />
<punto x="800" y="1500" />
</obstaculo>
</obstaculos>
  </mapa>
```

Fig 12. Resultado de la versión XML

El LectorMapa es un módulo que centraliza todas las representaciones. Las pruebas para el buen funcionamiento de la comunicación fue probado con la conexión de los distintos módulos que usan esas representaciones. Mediante un sistema de peticiones la potencia y posibilidad de ampliación de este módulo es ilimitada.

1.2.7. Prueba de funcionamiento global

Aquí se muestra un pequeño test realizado en la integración de: Cerebro, consola, LectorMapa, cargador Xml y Planificador Global(Grafo Visibilidad + búsqueda A*). Los datos se muestran en formato texto en una consola.

El robot comienza en el [10,1] y sus objetivos son:

- entrada(inicio) [1,1]
- salida(fin) [23,39]

Vemos como se carga el controlador (Pipeline) y seguidamente los módulos

```
Cargando Pipeline...
cerebro
consola
lectormapa
planificador
salidaTexo
temporizador
Pipeline Corriendo...
Salida Texto STD: arrancado
Temporizador arrancado
Planificador arrancado
```

El módulo de Mapa lee la lista de objetivos:

```
//XML
Mapa: id=a1 ruta=../etc/mapa1obj.bmp pxcm=1
ObjetivoCargado: salida(fin) [23,39]
ObjetivoCargado: entrada(inicio) [1,1]
//LECTOR MAPA
Mapa: id=a1 ruta=../etc/mapa1obj.bmp pxcm=1
Objetivo: salida [23,39]
Objetivo: entrada [1,1]
```

Calculamos los obstáculos:

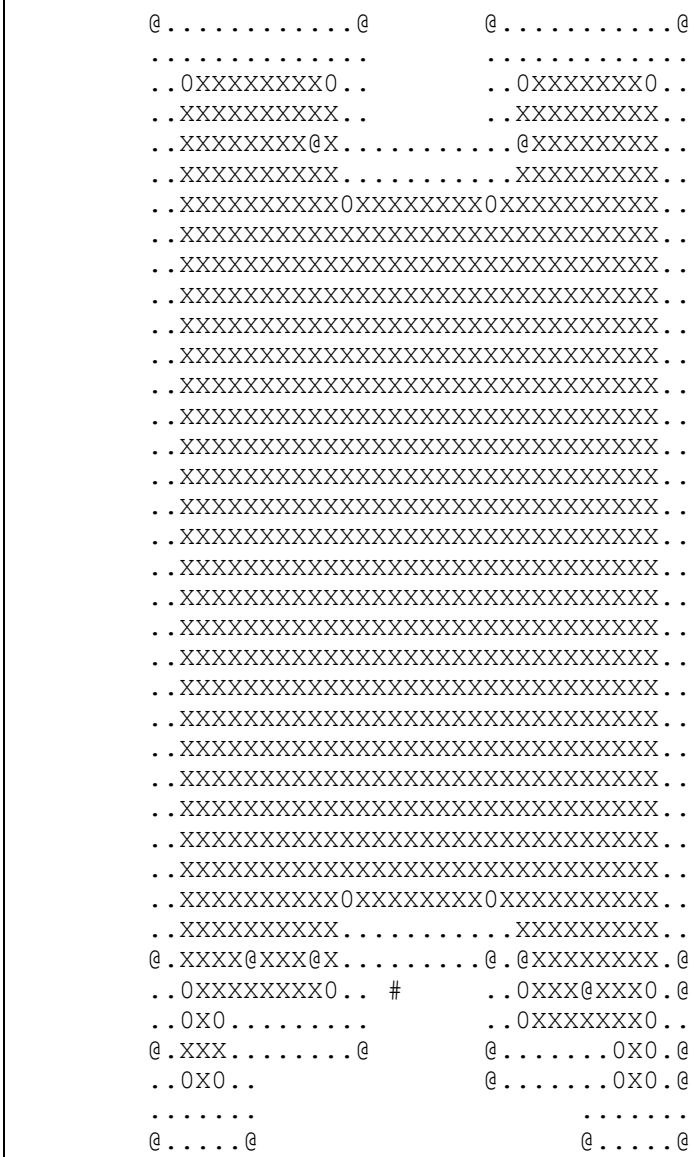
```
//CALCULO DE OBSTACULOS A PARTIR DEL BMP
Encontrado Obstaculo
a. x=10 y=9
b. x=19 y=9
c. x=10 y=39
d. x=19 y=39
Encontrado Obstaculo
a. x=31 y=9
b. x=39 y=9
c. x=31 y=39
d. x=39 y=39
Encontrado Obstaculo
a. x=20 y=13
b. x=29 y=13
c. x=20 y=36
d. x=29 y=36
Encontrado Obstaculo
a. x=10 y=40
b. x=12 y=40
c. x=10 y=42
d. x=12 y=42
Encontrado Obstaculo
a. x=31 y=40
b. x=39 y=40
c. x=31 y=40
d. x=39 y=40
Encontrado Obstaculo
a. x=37 y=41
b. x=39 y=41
c. x=37 y=42
d. x=39 y=42

Leer Configuracion Mapa Terminada: imagen=0x9dd390 ancho=50
alto=50
Mapa[50,50]
```

Enrejado como representación del mapa:

- X - ocupado
- @ - Engrosado
- - Esquinas
- # - Inicio y Final

```
// MAPA EN MODO ZONE (array de ocupado y libre)
 #
```



```
Consola STD: arrancado
LectorMapa arrancado
Cerebro arrancado
// Consola
Tamanyo maximo del comando: 20
|------------------|
```

```
| Consola ROBOT 3p3 |
|-------------------|
```

El usuario pide al robot que realice una guía:

```
TRobot>guia
OK guia
Estado Establecido = 2
```

El Robot transiciona a modo Guia y envía objetivos:

```
CEREBRO: EJECUTANDO MODO GUIA
CEREBRO: Calculando Objetivos...
CEREBRO: MODO NAVEGACION CON MAPA...
CEREBRO: Planificacion Lanzada...
Estado Establecido = 3

CEREBRO: EJECUTANDO MODO NAVEGACION
```

El Planificador procesa la trayectoria Punto Origen [10,1] Destino [1,1]

- Coste 9 unidades y mandamos los puntos consigna.
- Nodos intermedios ninguno.

```
PLANIFICADOR: ProcesandoTrayectoria, Ini=[10,1] , Fin=[1,1]
Grafo Aristas: 146
Grafo Nodos: 26
--------- A* ---------
num_edges: 146
num_nodos: 26
Start vertex: 0
Goal vertex: 1
Shortest path from 0 to 1: 0 -> 1
Total travel distance: 9
Planificador: Camino Encontrado Coste=9 -> MandandoTareas
FIN BUSQUEDA

CEREBRO: EJECUTANDO MODO NAVEGACION
```

Se procesa la siguiente trayectoria desde el [1,1] `al` [23,39] que es el final:

- Coste 50.9 unidades
- Nodos Intermedios 3 y 12

```
PLANIFICADOR: ProcesandoTrayectoria, Ini=[1,1] , Fin=[23,39]
Grafo Aristas: 152
Grafo Nodos: 26
--------- A* ---------
num_edges: 152
num_nodos: 26
Start vertex: 0
Goal vertex: 1
Shortest path from 0 to 1: 0 -> 3 -> 12 -> 1
Total travel distance: 50.9589
Planificador: Camino Encontrado Coste=50.9589 -> MandandoTareas
FIN BUSQUEDA

CEREBRO: EJECUTANDO MODO NAVEGACION
```

Se llega al destino y el cerebro termina la guía:

```
CEREBRO: GUIA TERMINADA
Estado Establecido = 1
```

El usuario pregunta el estado del robot y este responde IDLE:

```
TRobot>estado
CEREBRO: [Estado=1]
OK estado

TRobot>
CEREBRO: [Estado=1]
OK estado

TRobot>
```

1.2.8. Sonido

Inicialmente el módulo de sonido fue probado de forma aislada, comprobando que realizaba las operaciones básicas que queríamos obtener: cambiar el volumen de reproducción, admitir varios formatos de archivos de sonido…

Tuvimos problemas con la eficiencia del módulo puesto que consumía un gran porcentaje de la capacidad de procesamiento del

sistema, lo que resultaba poco deseable en un sistema de procesamiento en tiempo real como el nuestro. Después de varias pruebas conseguimos darnos cuenta de que el problema era debido a la mala utilización de "streaming" como método de reproducción de archivos de sonido. El muestreo del audio mediante streaming nos venía muy bien puesto que nos ahorra mucho el uso de memoria RAM necesario para la descompresión del sonido en su reproducción, pero por otro lado requiere un uso más intensivo de CPU puesto que la descompresión del archivo se realiza por partes (que son cargadas en un buffer) en vez de por completo en tiempo de carga. Pudimos arreglar el problema aumentando el tamaño del buffer de carga, consiguiendo así un equilibrio entre el consumo de memoria y el consumo de CPU por parte del módulo.

Mostramos a continuación una captura de pantalla de la ejecución del módulo aisladamente después de corregir el error comentado, se puede observar que el consumo de CPU es muy aceptable (1.09 %).

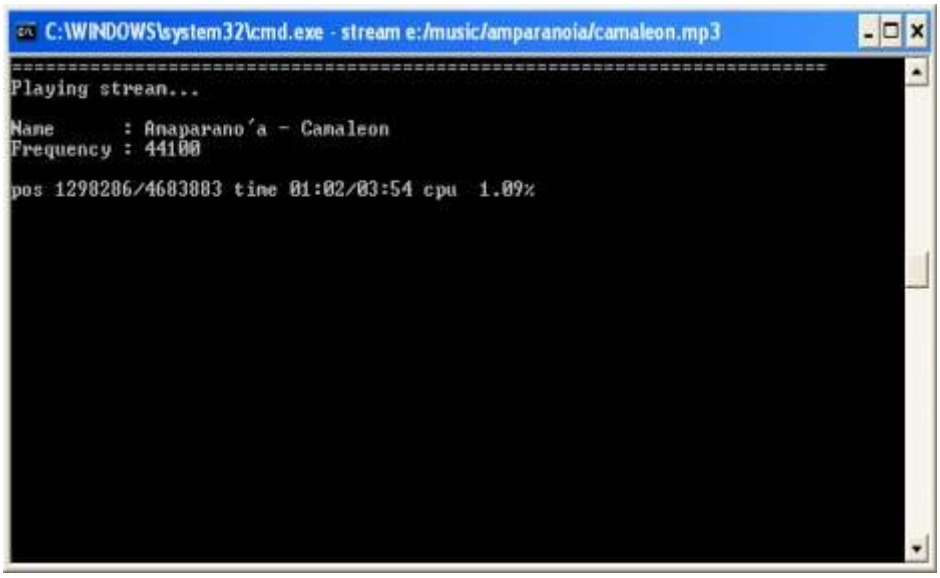
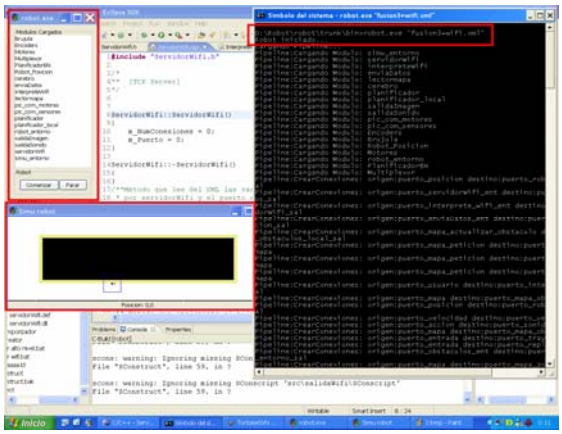


Fig 13. Captura ejecución aislada del módulo de sonido

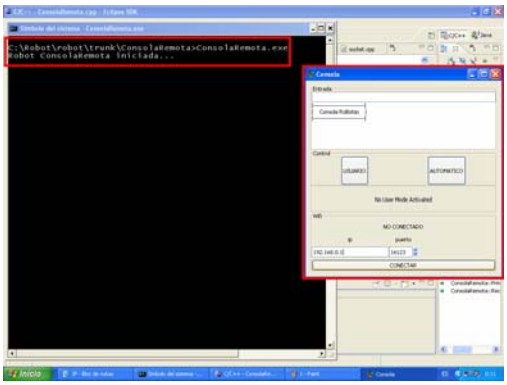
1.2.9. Conexión Wireless

A continuación vamos a mostrar algunas de las pruebas realizadas para el sistema de conexión y sus resultados. Mostraremos una secuencia de ejecución donde se puede comprobar su funcionalidad.

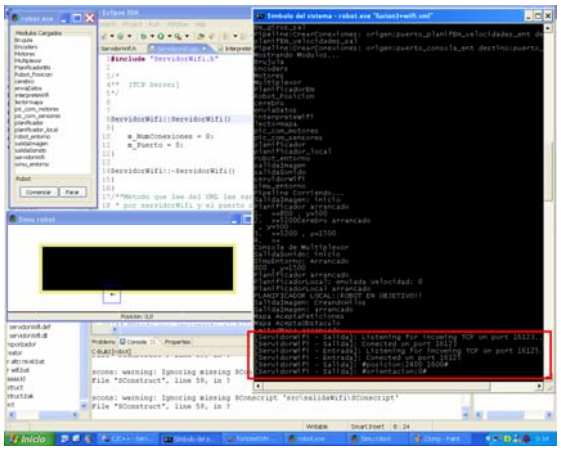
Ordenador con el Robot cargado.

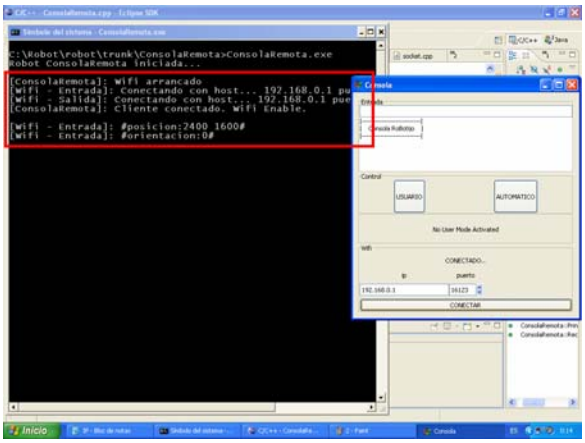


Ordenador con la Consola Remota cargada.

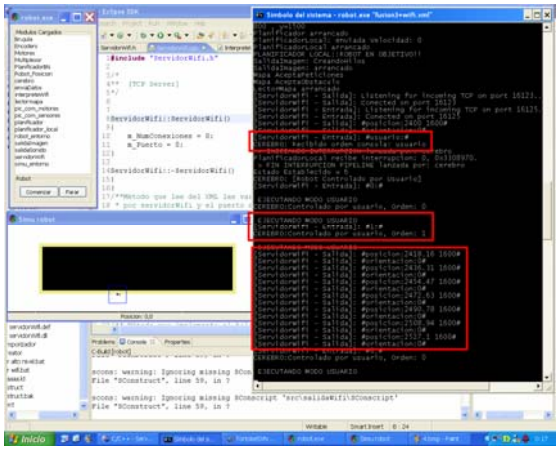


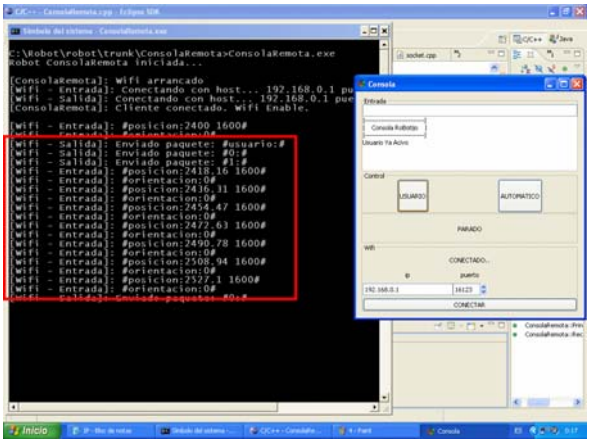
En el primer paso, vemos como se cargan los dos programas. El Robot en la imagen de la izquierda y la Consola Remota en la imagen de la derecha. Ambos ordenadores están conectados mediante una red Wifi y tienen una IP conocida.



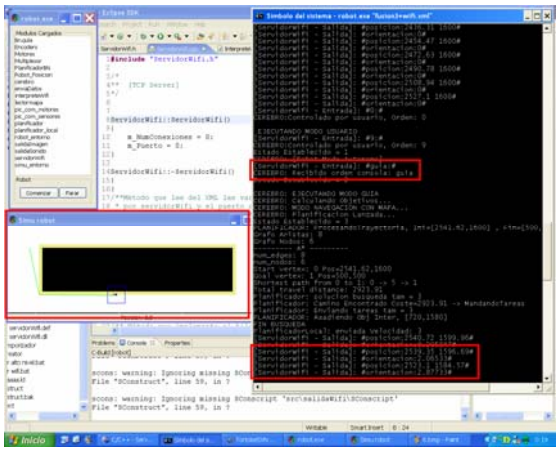


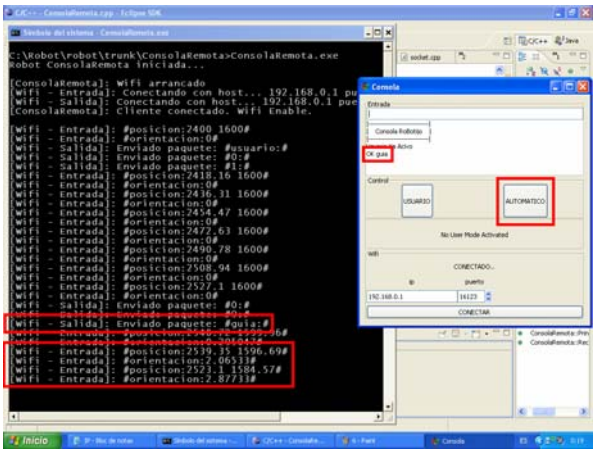
En el segundo paso podemos ver cómo el robot se pone a la espera de recibir una conexión. Al presionar sobre el botón conectar de la consola remota, se produce tal conexión. Una vez hecho esto, el robot manda su posición y orientación a la consola remota.



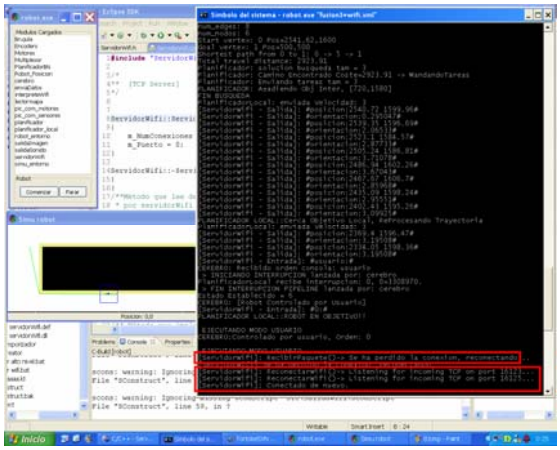


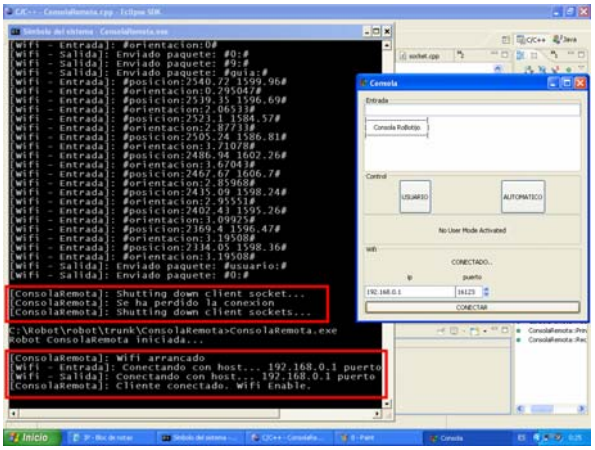
En el tercer paro podemos ver qué ocurre al presionar sobre el botón de usuario de la consola remota. Se manda la orden de usuario, que es interpretada por el módulo interpreteWifi, que la manda al cerebro para que este actualice su estado. Una vez en modo usuario, podemos mover el robot a nuestro antojo, ordenes que son enviadas con formato #0:# (parado), #1:# (avanzar), etc. Y que una vez interpretadas, hacen moverse al robot. Al moverse el robot, el módulo enviaDatos recoge las nuevas posiciones y orientaciones, que son enviadas a la consola remota, como podemos comprobar.





Si la orden, por el contrario es que ejecute un comando en modo automático, por ejemplo, guía, podemos comprobar que el comportamiento de la conexión es el mismo.





Por último, podemos que ver que en caso de que se pierda la conexión o de que se cierre la consola remota, el robot pone automáticamente los puertos a la escucha de una nueva conexión. Esta nueva conexión se puede realizar de forma automática por parte de la consola remota o abriendo una nueva consola remota en caso de que ésta hubiera sido cerrada. Esto quiere decir que el robot puede funcionar sin necesidad de una consola remota que lo monitorice, pudiendo iniciar ésta en cualquier momento.

1.2.10. Apartado gráfico

Hemos usado unas librerías portables Glibmm(ref[18]) y GTKmm, que han dado multitud de problemas par ejecutar varias ventanas a la vez. Una vez arreglado el problema, nos dimos cuenta que el rendimiento bajo windows no es óptimo ya que la modificación de un elemento de la ventana cada 100 milisegundos, llega a bloquear el refresco de las ventanas. Por ello decidimos realizar solo refrescos rápidos sobre ventanas preparadas para gráficos (GDK) y no refrescar elementos básicos de la ventana tal como labels o botones.

2. Integración

2.1.Introducción

La integración era una de las partes más cruciales del Proyecto. Para eso habíamos dedicado mucho tiempo a pensar bien el diseño de nuestro sistema. Y después de ver los resultados, vimos que el diseño había sido el acertado, ya que la unión o sustitución de módulos era instantánea. Esto ha sido uno de los puntos más fuertes de nuestro proceso de desarrollo.

2.2.Integración por partes

2.2.1. Integración software alto nivel y bajo nivel

Para la integración con la parte de bajo nivel se realizaron los siguientes pasos:

Simular la parte de bajo nivel con módulos que según el diseño, cumplían las mismas características y tenían las mismas entradas y salidas:

- Simulación Robot: Modelo del robot simplificado al control de la posición y la orientación.
- Simulación Entorno: Modelo muy simplificado de la aparición de obstáculos en el camino.

Pruebas del conjunto: Retocamos los modelos hasta que vimos que se comportaban como queríamos y que se parecían al modelo real.

Sustitución o reconexión de los puertos con la definición de las entradas de la parte de bajo nivel.

2.2.2. Integración final con el robot físico (Hardware)

Se realizó un módulo de comunicación con el microcontrolador pic, que era el nexo de unión con el Hardware real. Previamente se unió a la parte de bajo nivel software, y posteriormente a la de alto nivel. El resultado a bajo nivel fue satisfactorio, pero en conjunto hubo problemas de pérdida de información con respecto al pic. Llegamos a la conclusión de que era un desajuste en la lectura del puerto serie, por la falta de control de flujo. Esto se debía a la emulación del puerto serie sobre una conexión usb, y el driver utilizado.

La solución era crear un driver de comunicación seguro, pero no era el objetivo del proyecto ni había tiempo incluido en la planificación para ello. Así que decidimos realizar un nuevo módulo de alto nivel de comunicación con el puerto COM. Para ello implementamos una lectura y escritura en overlapping y un buffer seguro (con mutex) que iba almacenando los datos leídos. Finalmente por medio de operaciones sobre el buffer podíamos recoger el número de bytes que quisiéramos o incluso decidir un carácter de terminación y recoger hasta ese carácter.

Por medio de una conexión virtual entre puertos de comunicaciones y una aplicación de testing simulamos los datos enviados por el pic y veíamos lo que éste recibía. Cuando todo esto funcionaba pasamos a probarlo con el robot físico. Para variar, hubo que hacer distintos ajustes en los parámetros de configuración del robot (distancia de seguridad, distancia de los puntos que mandaba el planificador al control de bajo nivel,..) ya que con la simulación la comunicación era ideal.

Muchos de los datos enviados eran corruptos, así que optamos por la redundancia de los datos, con lo que siempre alguno llegaba bien. En la parte II de este documento, sección módulo Pic_Com, está toda la información detallada acerca de este módulo.

2.3.Pruebas en conjunto después de la integración

Pasamos a enumerar ahora el proceso de pruebas y resultados de los grandes bloques del sistema desarrollado después de su integración.

2.3.1. Control Remoto

Nada más ejecutarlo observamos que funcionaba, de hecho existen sendos videos que lo acreditan en el cd que se suministra junto con esta memoria.

2.3.2. Guía Ciega

La guía ciega es aquella que se hace sin sensores, es decir el robot no utiliza el planificador local, nada más que para dividir la trayectoria. Estas guías son muy útiles para el ajuste de los parámetros y comprobación del buen comportamiento del robot.

Seguimos los siguientes pasos:

1. Realizamos diversos mapas de prueba, y para probarlos en un inicio lo simulamos, viendo como se generaban las trayectorias.

2. Cuando el mapa pasaba los test, pasamos a ejecutarlos en robot usando un soporte para que el robot estuviera en alto y no hubiese problemas de colisiones.

3. Si no había ningún incidente realizábamos la prueba sobre el suelo en un espacio amplio, ajustando los parámetros

El siguiente diagrama explica este proceso.

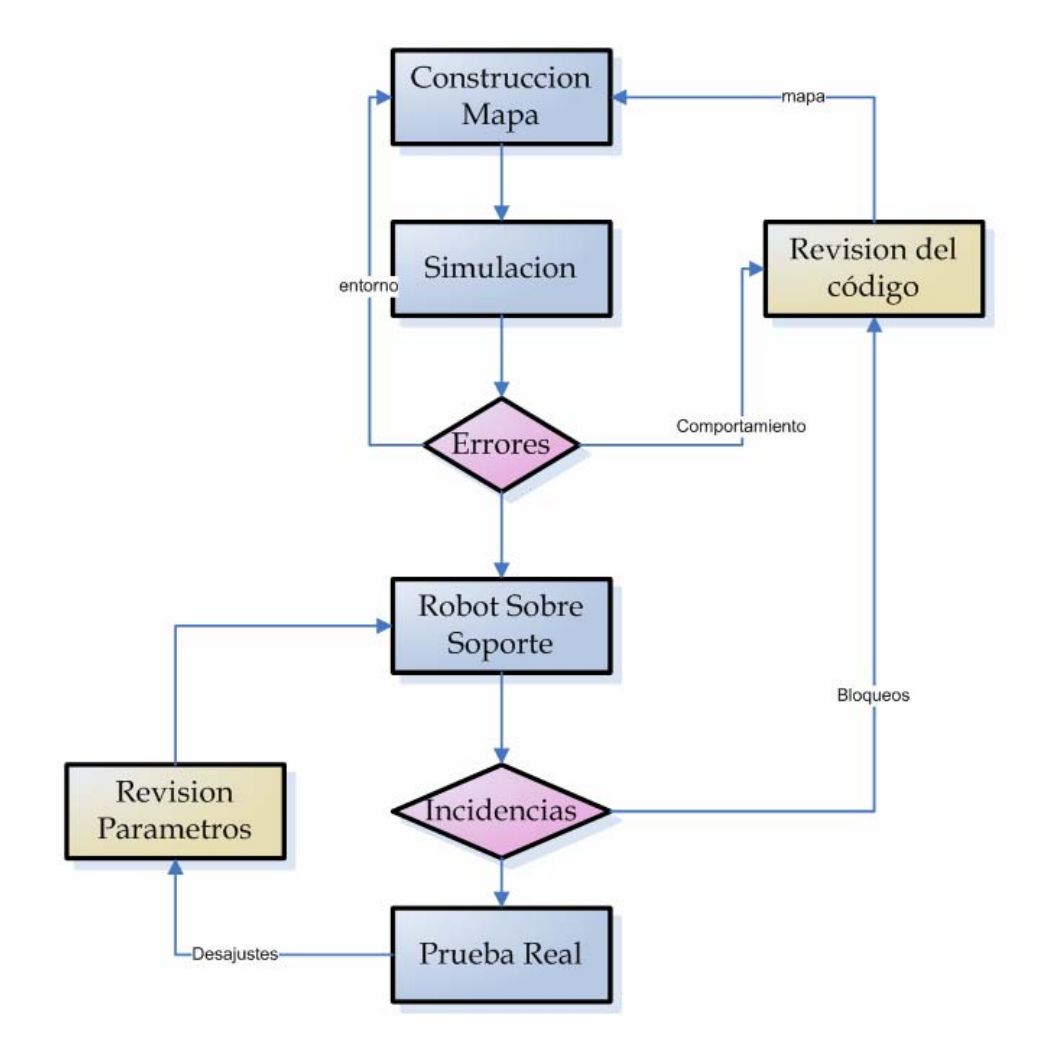


Fig 14. Diagrama de pruebas de guía ciega

2.3.3. Guía Real

La guía real utiliza la información de los sensores como medida de ayuda anti-colisión. Así en muchos casos el planificador local tomará el control y hará caso omiso de las trayectorias generadas globalmente.

Para la guía real comprobamos el funcionamiento de los sensores y de los motores por separado. Luego simulamos el comportamiento del robot con sensores simulados, incluyendo obstáculos aleatorios en el entorno. Finalmente con el robot físico, realizamos varias pruebas con obstáculos estáticos simulados y objetos móviles simulados.

La integración de los sensores y los motores, contaron con muchos problemas técnicos por errores de las placas de los pic, y el funcionamiento de Windows bajo los drivers que controlan el pic.

3. Trabajo futuro

Puesto que hemos utilizado una arquitectura modular, el sistema que hemos implementado puede crecer fácilmente añadiendo nuevas funcionalidades y mejoras al robot final que hemos obtenido. Mostramos a continuación algunas de las posibles extensiones futuras al proyecto.

3.1. Uso de balizas para la localización del robot

Uno de los objetivos a los que no se ha podido llegar este año ha sido la utilización de balizas que guiaran el sistema de localización del robot. Hemos tenido que conformarnos con usar la información proporcionada por los encoders, que resulta mucho menos precisa y que acumula mucho error.

Las causas por las que no hemos podido cumplir este objetivo han sido problemas administrativos con la compra del material, que en un principio si estaba pensado adquirir, y también problemas debidos a la organización física del museo que hace difícil la colocación de este tipo de dispositivos de localización.

Por todo esto, una ampliación esencial al trabajo de este año sería llevar a cabo este sistema de localización, lo que lleva implícito el desarrollo del software necesario para ubicar al robot en su entorno, interactuando con la información suministrada por las balizas.

3.2. Visión artificial

Una ampliación muy interesante sería dotar a nuestro robot de un sistema de visión artificial, que permitiría al robot realizar todo tipo de tareas nuevas.

Se podrían crear mapas dinámicos de las distintas escenas por las que se moviese el robot y utilizar éstos para guiar su navegación. Además el robot podría retransmitir en tiempo real a cualquier equipo de la facultad imágenes de su recorrido, para poder tener una monitorización sobre él.

También se podría desarrollar un sistema de reconocimiento de gestos ó de carteles según los cuales el robot pudiese identificar la acción a realizar (explicación de una pieza concreta del museo, retroceder, informar mediante la wifi de la llegada de algún visitante...).

Asimismo se podrían ampliar las funciones del robot y además de guía del museo podría desarrollar las tareas de vigilante de la facultad. Podría detectar la apertura de alguna de las ventanas durante la noche o la presencia de algún intruso dentro de la facultad. Después mediante algún sistema de comunicación (wifi, gsm,...) mandaría la alarma correspondiente al vigilante de seguridad humano que se encontrase más cercano al edificio.

3.3. Nuevos sistemas de control del robot

Nuestro robot puede controlarse actualmente mediante el propio equipo informático que tiene instalada nuestra aplicación y remotamente mediante cualquier otro equipo usando una consola remota y una conexión wifi. Estos sistemas se podrían ampliar por ejemplo con la posibilidad de manejar y monitorizar el control del robot mediante dispositivos portátiles como PDAs o teléfonos móviles de última generación. También se le podría añadir una pantalla táctil a la plataforma robótica a través de la cual se podría interactuar con el robot.

Fig 15. Uso de una PDA para controlar el robot

3.4.Mejoras en el apartado gráfico

A lo largo del proyecto y conjuntamente con el cumplimiento del objetivo principal del mismo (conseguir un robot guía autónomo), hemos desarrollado un entorno 2D para visualizar una simulación del recorrido del robot en cada misión. Este entorno resulta muy básico y no demasiado atractivo, una ampliación interesante a esta parte del proyecto consistiría en desarrollar un entorno 3D de simulación más elaborado, con tratamiento de texturas, renderizado de imágenes... . Se podría incluso, intentar conseguir una recreación fidedigna del museo por el que se mueve el robot, mediante imágenes fotográficas tridimensionales. Esto constituiría un atractivo muy importante de la aplicación y facilitaría las funciones de monitorización del robot.

Fig 16. Ejemplo de simulación 3D del entorno del museo mediante fotografías.

3.5.Mejora del módulo cerebro

3.5.1. Definición Externa de la máquina

Por medio de un archivo externo, por ejemplo XML, se puede definir la máquina de estados y las acciones asociadas, pudiendo realizar desde la máquina de estados más simple a la más compleja sin compilar. Esta función no esta implementada del todo, pero solo con incluir un lector XML que llame a las funciones CrearEstado, AñadirTransición, estaría implementado. No lo hemos realizado porque en nuestro caso es una aplicación muy específica. Es decir que la máquina siempre es la misma.

3.5.2. Ampliación al scripting o lenguaje alto-alto-nivel + Yacc y Lex

La potencia del código es bastante pequeña si tenemos en cuenta que hay que reimplementar y compilar cada acción si queremos cambiar el comportamiento del módulo cerebro. Para eso se ha incluido una tabla de acciones, o pares (identificador, función asociada) que simplemente registrando las funciones en el programa se puede usar scripting, o implementando un módulo por encima del cerebro, que compile sentencias de muy alto nivel para modificar los comportamientos. Esa posibilidad se ha dejado, pero las funciones se han implementado fijas, ya que nuestro proyecto se centraba en un robot guía, y no había que modificarlo. Aún así se deja a los futuros colaboradores la opción de llevarlo a modo script, pudiendo usar el robot en cualquier tarea, tan solo cambiando una función.

Por ejemplo modificando la acción asociada al estado navegación con mapa para que elija un objetivo aleatorio en vez de que siga un orden de llegada.

PARTE II

Detalle de los módulos

Capitulo 4

Núcleo del sistema

1. Introducción

En este apartado se ofrece una visión detallada de los módulos que forman el núcleo central del sistema que hemos desarrollado. Se trata de los componentes que realizan el control efectivo a alto nivel de nuestro robot, formando un subsistema suficiente para cumplir el objetivo principal de nuestro proyecto, la resolución del problema de la planificación de trayectorias en un robot móvil autónomo. Se puede decir que este subsistema representa la "inteligencia" y la "visión" de nuestro robot.

Posteriormente en el siguiente capítulo incluimos la descripción del resto de módulos que conforman el sistema global, y que están encaminados a la comunicación de entrada salida con el robot.

2. Módulo Cerebro

2.1.Introducción

Es el módulo de más alto nivel y quien tiene la visión más completa del mundo. Por ello su potencia es mucho mayor que su uso real.

Según las últimas tendencias en Inteligencia Artificial con entes móviles, ya sean robots, agentes o personajes de videojuegos, el método empleado es las máquinas de estados finitos asociadas a búsquedas en tiempo real (ref [5]). Es curioso que estructura tan simple simule la inteligencia artificial, pero también hay que tener en cuenta que para entes reactivos a un entorno es muy buen modelo. Al igual que en la naturaleza, a partir de un estado e1, con una entradas x1…xn, se provoca una reacción que modifica el estado, pasando a un estado e2 con unas acciones a1…an,

y así sucesivamente. De hecho en programación de agentes, y en robots colaboradores, da muy buenos resultados. Por ello hemos decidido implementar una máquina de estados que coordine el cerebro.

2.2.Detalles

Entrada: Ordenes de Usuario, Información de alto nivel del robot: Posición y Velocidad

Salida: Modo automático: Una trayectoria con estructura(Punto en el espacio origen, Punto en el espacio destino). Modo usuario: ordenes codificadas de movimientos de control remoto(Parar, Adelante, Atrás…)

Descripción: Módulo encargado de tomar las decisiones y controlar todo lo que está por debajo de él. Es la Pseudo-Inteligencia del Robot.

2.3.Arquitectura y funcionamiento del módulo

2.3.1. La Máquina de Estados

Es el control interno del cerebro y es reactivo a estímulos procesados en forma de transiciones.

En la definición de la máquina de estados se decide qué transiciones aceptan cada estado y a qué estado pasan.

A continuación veremos los componentes de la máquina de estados

2.3.1.1. Estados

- Pares (Transición, Estado Siguiente)
- Acción que se ejecuta en ese estado
- Acción asociada a la transición

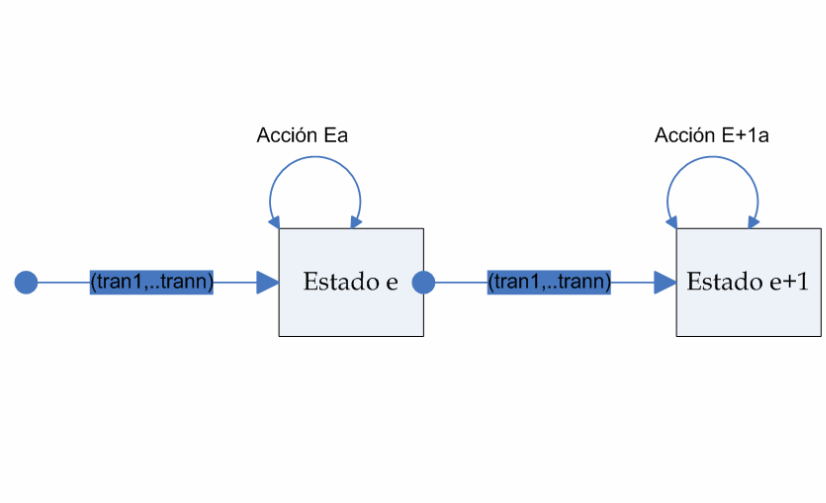


Fig 17. Modelo estado-transición de cerebro

2.3.1.2. Tabla de Acciones

Es el mapa que define las acciones asociadas a cada estado y a cada transición.

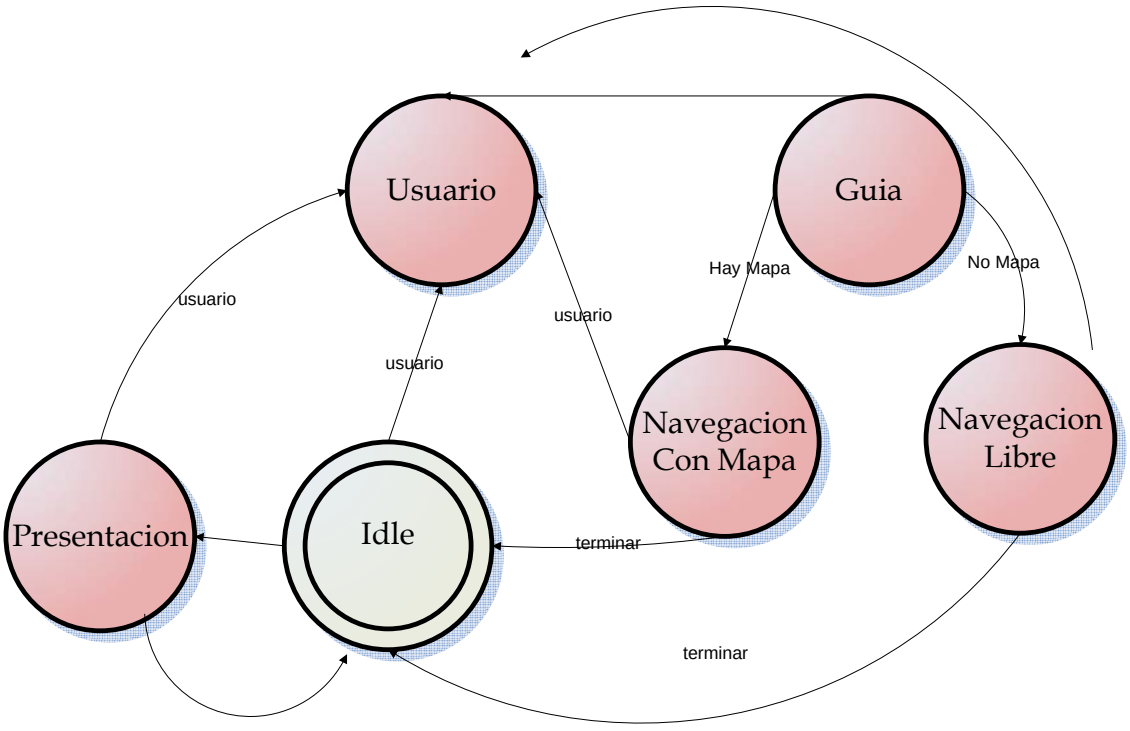


Fig 18. Maquina de estados del Robot

2.3.2. Modos de Cerebro

El cerebro tiene dos modos de funcionamiento:

Automático: El robot se desenvuelve solo y aunque pueda recibir ordenes de usuario por la entrada, él decide que acciones realiza, y en consecuencia como se mueve. Este modo es clave para realizar guías.

Usuario: El Cerebro cede el control al usuario, con lo que manda las peticiones directamente a bajo-nivel, para que mueva el robot. Este modo es ideal para el control teledirigido. Aún así el cerebro puede orientar a la persona que lo esta controlando o restringir acciones.

Cuando se pasa a modo usuario el estado de la máquina de estados es *usuario*. Y cuando es automático el estado por defecto es *idle*.

2.3.3. Acciones asociadas a cada estado

Mostramos a continuación las acciones asociadas a cada estado de la máquina de estados del módulo cerebro.

2.3.3.1. Idle

El robot esta en espera de transiciones y no realiza ninguna acción.

2.3.3.2. Usuario

El cerebro acepta ordenes directas de movimiento y las manda a bajo nivel, para que las ejecute.

2.3.3.3. Guía

El robot en modo autónomo puede entrar en guía, por petición expresa del usuario o por si mismo.

Se leen los objetivos del xml. Se comprueba si el robot ya está en el objetivo, y en tal caso se avanza hacia el siguiente objetivo. Si existen y no hay ningún error, se pasa a modo navegación con mapa, y sino a navegación libre.

2.3.3.4. Navegación con mapa

Es la acción más compleja y tiene que conocer la posición y la velocidad del robot. También hay que tener en cuenta estos dos conceptos:

Tiempo anterior: tiempo que define cuando se ejecuta la acción asociada al objetivo. Para tiempos positivos la acción se realiza t seg. después de haber llegado al objetivo. Para tiempos negativos, la acción se realiza t seg. antes de llegar al objetivo.

Tiempo de salida: Define el tiempo de espera desde que se ejecutó la acción hasta ponerse en marcha hasta el siguiente objetivo.

```
Pseudo código:
Si Comienzo → Nueva Trayectoria
Si Acción antes de llegar (t negativo)
      Estimar tiempo de llegada
      Si cumple el tiempo →
            Lanzar Acción asociada a objetivo
            Recibir Tiempo de sincronización. (duración)
      Si  Robot en Objetivo →
            Si Ejecutando Acción →
                  Tespera = Tsyncro – Tpasado + Tsalida
                  Esperar
            Sino →
                  Lanzar Acción asociada a objetivo
                  Sincronizar
                  Esperar
      Si ¬UltimoObjetivo → NuevaTrayectoria
      Sino → Terminar
```

2.3.3.5. Navegación Libre

Esta acción no esta implementada.

2.3.3.6. Presentación

El cerebro realiza una pequeña demostración configurable, en la cual se mueve y habla a la vez.

3. Módulo Temporizador

3.1.Introducción

Para una aplicación en tiempo real, en concreto un robot móvil, es muy importante el control del tiempo. Además es útil y dota de potencia a la aplicación software que éste sea genérico y fácil de usar. Para ello hemos creado un servicio en forma de módulo que se encarga de controlar el tiempo.

Veamos algunos ejemplos de utilidad para hacernos a la idea:

- El cerebro esta en Idle, y queremos que cada cierto tiempo compruebe que no tiene a nadie delante. Pues simplemente se hace una petición al timer de ejecución infinita que avisará cada x tiempo.

- El cerebro esta en modo navegación pero hace tiempo que no se mueve, posiblemente porque esta bloqueado. Se hace una petición de alarma de una sola ejecución como si fuera un time-out. Pasado ese tiempo el Temporizador avisa y el cerebro obra en consecuencia, pasando a estado Idle para no agotar recursos, o vuelve a hacer la guía.

Este módulo funciona dada la velocidad de la arquitectura, sino habría que crear un temporizador nativo dentro de la arquitectura. Aún así la exactitud de las alarmas no esta garantizada, con lo que no es aconsejable utilizarlo en tareas que requieren una precisión elevada, como módulos relacionados con bajo nivel.

3.2.Detalles

Entrada: Peticiones de Alarma o Timers.

Salida: Alarmas o Avisos, con el modulo destino asociado.

Descripción: Módulo encargado del control del tiempo de forma centralizada.

3.3.Arquitectura y funcionamiento

El funcionamiento externo del módulo, es decir que otros programadores deben saber para usar este módulo es el siguiente:

Se lee la petición del puerto. Dependiendo del tipo de petición se crea un timer diferente. Cada vez que se cumpla el tiempo pedido se mandará una señal al puerto de salida relacionado con el timer. Hasta que se cumplan el número de ejecuciones pedidas (Petición con n=0 es un timer infinito).

3.3.1. Funcionamiento Interno

El módulo contiene dos tablas:

- Tabla de (timer, TimeoutHandler): Se asocia cada timer representados como enteros, con una función que despierta cada t tiempo transcurrido y ejecuta una función de callback, que será la acción asociada a la petición.

- Tabla de (timer, Número Ejecuciones): Indica cuantas ejecuciones se deben realizar todavía y cuando eliminar el Timer del sistema.

Cuando llega una petición se crea y se añade un timer a la tabla y cada t (tiempo pedido) el hilo de TimoutHandler despertará, y con los parámetros de la función ejecutará la función de callback asociada. Decrementará en 1 el número de ejecuciones si esta no es 0 (infinita) y en caso contrario eliminará el timer de la tabla y todos los datos asociados.

Para entender el comportamiento hay que tener tres conceptos claros en la cabeza, el hilo dormido, la señal para despertar y la función de callback. Aquí explicaremos por encima su funcionamiento y para más información véase la documentación Glibmm (ref[18]). La creación del timer se basa en la creación de una función dinámicamente y asignarle una posición de memoria. Como parámetros se le dan la función de callback y el id del timer. Luego se asocia a esta función creada una señal de time_out con el tiempo de la petición. Por último se añade a las tablas para poder tener control sobre él para su posterior eliminación o modificación.

4. Módulo Planificador

4.1.Introducción

Nuestro robot como cualquier robot móvil autónomo se caracteriza por una conexión inteligente entre las operaciones de percepción y acción, que definen su comportamiento y le permite llegar a la consecución de los objetivos programados sobre entornos con cierta incertidumbre. El grado de autonomía depende en gran medida de la facultad del robot para abstraer el entorno y convertir la información obtenida en órdenes, de tal modo que, aplicadas sobre los actuadores del sistema de locomoción, garantice la realización eficaz de su tarea. De este modo, las dos grandes características que lo alejan de cualquier otro tipo de vehículo se relacionan a continuación (ref [10]):

- Percepción: Determina la relación del robot con su entorno de trabajo, mediante el uso de los sensores de a bordo.

- Razonamiento: Determina las acciones que se han de realizar en cada momento, según el estado del robot y su entorno, para alcanzar las metas asignadas.

De este modo, la capacidad de razonamiento del robot autónomo móvil se traduce en la planificación de unas trayectorias seguras que le permitan la consecución de los objetivos encomendados. La ejecución de la tarea debe realizarla en bucle cerrado para adaptarse a la navegación sobre entornos parcialmente desconocidos (pueden aparecer obstáculos de forma imprevista). No se emplea un bucle de control tradicional, ya que la acción no se genera por la simple realimentación de la salida.

Por tanto, resulta necesario el uso de un planificador con capacidad de análisis geométrico que conozca el estado del entorno y del robot junto a sus características cinemáticas y dinámicas. De este modo, se realiza la transformación de los datos suministrados por la percepción en referencias de control adecuadas, y que definan una trayectoria libre de obstáculos que garantice el logro de las metas determinadas en la tarea.

Así, el planificador se configura como el responsable, en gran medida, de la eficacia del vehículo en la navegación, por lo que en consecuencia, su diseño precisa una especial atención.

4.2.Formalización del problema de la planificación

El entorno de trabajo en el cual un robot móvil realizará su tarea, puede considerarse como un conjunto de configuraciones (ref [10]) en las cuales puede encontrarse el robot en un determinado instante de tiempo. Entre ellas existirá un subconjunto inalcanzable, al estar ocupado por los obstáculos del entorno.

Se define una configuración q de un robot como un vector cuyas componentes proporcionan información completa sobre el estado actual del mismo. Un robot es un objeto rígido al cual se le puede asociar un sistema de coordenadas móvil. La localización del vehículo en un determinado instante de tiempo queda definido por la relación existente entre el sistema de coordenadas global F_g en virtud del cual está definido todo el entorno de trabajo y su sistema de coordenadas locales asociado F_r.(Fig 19.).

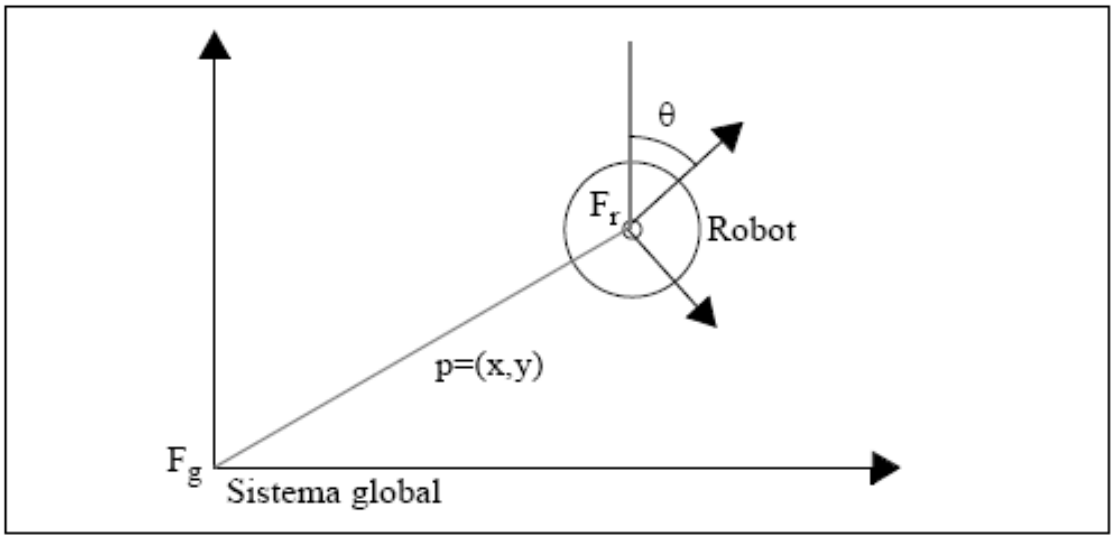


Fig 19. Sistema de coordenadas global, y sistema local asociado al robot

El vector que proporciona información sobre el estado actual del robot viene dado, en principio, por dos componentes: la posición p y la orientación θ. Por tanto, se puede definir configuración como:

$$q = (p, \theta) = (x, y, \theta)$$

Se denomina espacio de configuraciones *C* del robot *R* a todas las configuraciones *q* que puede tomar el robot en su entorno de trabajo. El subconjunto de *C* ocupado por el robot *R* cuando este se encuentra en *q*, se denota por *R(q)*. Si el robot se modela de forma circular con radio ρ, *R(q)* se define como:

$$R(q) = \{ q_i \in C / || q, qi|| \leq \rho \}$$

En el espacio de trabajo, donde el robot realizará su tarea, se encuentran distribuidos una serie de obstáculos definidos como un conjunto de objetos rígidos *B* y que se encuentran distribuidos por el espacio de configuraciones *C*.

$$B = \{b_1, b_2, ..., b_q\}$$

El conjunto de configuraciones del espacio *C* ocupadas por un obstáculo se define por $b_i(q)$, de tal forma que el subconjunto de configuraciones de *C*, que especifican el espacio libre de obstáculos viene dado por:

$$(\text{exp. 1.})\ C\ell = \{ q \in C / R(q) \cap \left(\bigcup_{i=1}^{q} bi(q) \right) = \phi \}$$

Según esta metodología, el problema de la planificación, tal y como se ha definido, queda transformado en la búsqueda de una sucesión de posturas *q* tal que la primera de ellas sea la postura actual del robot q_a y la última de esta sucesión la postura objetivo q_f.

Todas las posturas de la serie deben pertenecer al subconjunto $C\ell$ definido en exp. 1.. Es decir, una ruta Q_r que conecta la postura inicial q_a con la final q_f es:

$$Q_r = \{ q_a, \ldots, q_f / q_i \in C\ell \}$$

La especificación de este conjunto Q_r, implica la construcción de una función ruta definida de la siguiente forma:

$$(\text{exp. 2.})\ \tau: [0,1] \rightarrow C\ell$$

tal que:

$$\tau(0) = q_a \quad \tau(1) = q_f$$

Además de la restricción mostrada en exp. 2., se le exige a la función τ la continuidad. Este concepto se refleja en la siguiente expresión:

$$\lim_{s \rightarrow s_0} ||\tau(s), \tau(s_0)|| = 0$$

4.3.Planificador específico de nuestro sistema

4.3.1. Introducción

El esquema presentado contiene en líneas generales el diagrama de flujo de información entre las distintas fases de las que se compone la navegación de nuestro robot. No se trata de un diagrama de diseño del sistema (el cual se puede consultar en la parte I de este documento) si no de un esquema para comprender mejor las distintas tareas que se presentan en la navegación de un robot móvil autónomo.

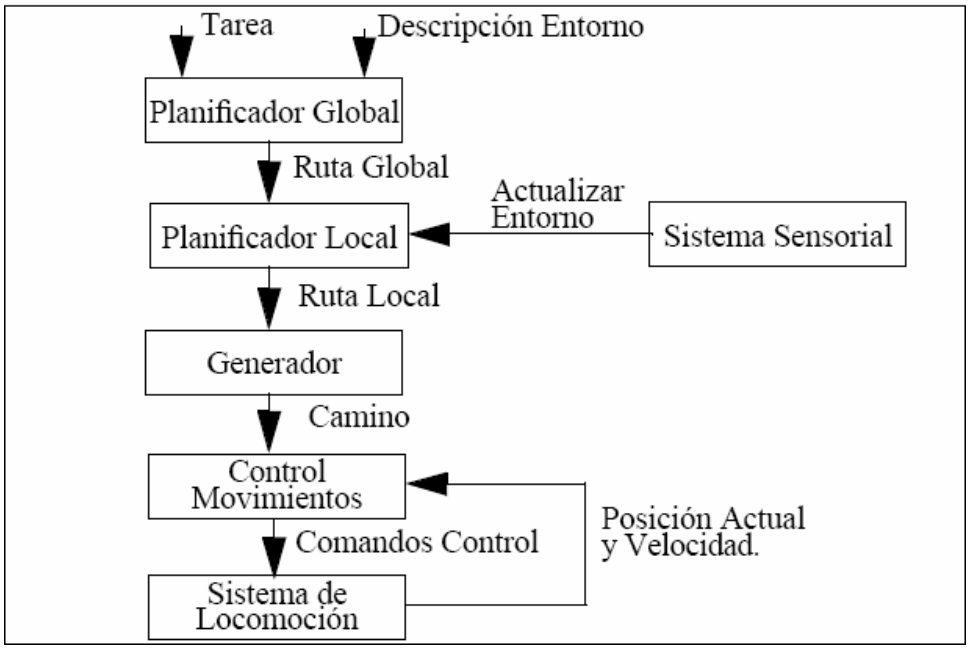


Fig 20. Flujo de datos de los planificadores.

Como puede observarse hemos dividido la tarea de planificación en dos partes: planificación global y local. La primera de estas subtareas construye una ruta sobre la cual se puede definir un camino libre de obstáculos según la información que a priori se posee del entorno. Si la descripción del entorno introducida fuese perfecta, la ruta calculada sería de forma directa la entrada de la tarea generador de bajo nivel. Sin embargo, al no serlo, puede dar lugar a la construcción de un camino que no esté libre de obstáculos, con el consiguiente peligro de que el vehículo impacte con algún elemento del entorno.

La tarea de planificación local recibe información del sistema sensorial sobre el entorno local del robot, según el radio de alcance de los sensores externos de a bordo. Mediante el análisis de estos datos actualiza el modelo preliminar del entorno y decide si se precisa replanificar la ruta local del robot.

La clave del esquema presentado en la Fig 20. para adaptarse a diversos entornos, aunque no se posea un conocimiento exhaustivo del mismo, reside en la distinción efectuada entre planificación global y local. Ambos conceptos se pueden definir con mayor precisión de la forma que sigue (ref [9]):

Planificación global: Construir o planificar la ruta que lleve al robot a cada una de las submetas determinadas por el control de misión, según las especificaciones del problema que debe resolverse. Esta planificación es una aproximación al camino final que se va a seguir, ya que en la realización de esta acción no se consideran los detalles del entorno local al vehículo.

Planificación local: Resolver las obstrucciones sobre la ruta global en el entorno local al robot para determinar la ruta real que será seguida. El modelo del entorno local se construye mediante la fusión de la información proporcionada por los sensores externos del robot móvil. La construcción de la ruta global puede realizarse antes de que el vehículo comience a ejecutar la tarea, mientras que la planificación local se lleva a cabo en tiempo de ejecución. En el caso de realizar una navegación sobre entornos totalmente conocidos es obvio que resulta innecesario proceder a una planificación local, pero en nuestro caso aumenta la relevancia de la misma puesto que el conocimiento del museo de informática donde estará el robot no es completo, pueden aparecer personas que obstaculicen el paso del robot, carritos del servicio de limpieza del edificio...

En este apartado ofrecemos una visión detallada del subsistema planificador de nuestro proyecto, especificando en un principio los métodos de planificación que hemos utilizado para, a continuación describir los distintos módulos que integran este subsistema que son: búsqueda, planificador global y planificador local.

4.3.2. Métodos de planificación utilizados en nuestro planificador

Para resolver el problema de planificación de trayectorias que debe seguir nuestro robot hemos utilizado el método de grafos de visibilidad para guiar la planificación global y el método de enrejado de celdas para guiar la planificación local.

La descripción de éstos y otros métodos clásicos de planificación se encuentran detallados en el apéndice A de esta memoria.

4.3.3. Búsqueda

4.3.3.1. Introducción

La inteligencia en la planificación de caminos es la búsqueda heurística. No existe planificación global sin una buena búsqueda por detrás. Por eso le hemos dado en el proyecto una importancia destacada. Las operaciones de movimiento en tiempo real, cuando el robot funciona de manera autónoma, están muy influenciadas por la búsqueda. Por ejemplo: Planificación Global, Esquivar un Obstáculo.

Dentro de las búsquedas necesitamos dos tipos, según nuestro diseño del planificador: Búsqueda sobre Grafos (Planificación Global), y búsqueda sobre enrejado (Planificación Local).

4.3.3.2. Búsqueda sobre grafos

La búsqueda sobre grafos implementada se basa en la construcción estática de un grafo como lista de adyacencia, y una búsqueda A* con heurística de distancia euclídea.

4.3.3.2.1. Detalles

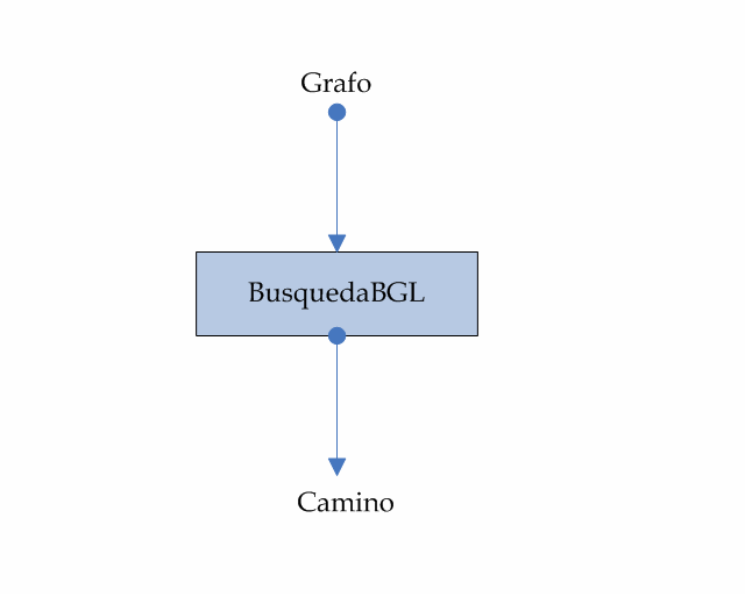


Fig 21. Entrada y salida de BusquedaBGL

Búsqueda sobre Grafos como lista de adyacencia:

Entrada: Grafo, Nodo Inicial, Nodo Final

Salida: Camino desde el nodo inicial, al nodo final.

Descripción: Búsqueda rápida A* sobre grafos que hace uso de algoritmos de la librería Boost de BGL. Por medio de templates y visitantes podemos reconfigurar a nuestro agrado el funcionamiento de la búsqueda.

Tipo de grafo: Grafo no dirigido representado como una lista de adyacencia.

Velocidad de reacción: Cercano a tiempo real para universos restringidos. Muy recomendable para planificaciones globales y replanificaciones sobre espacios grandes.

4.3.3.2.2. Requisitos

El grafo debe poseer:

- Lista de Nodos
- Lista de Aristas
- Lista de Costes
- Nº de Nodos
- Nº de Aristas

Por defecto el nodo inicial es el 0 y el nodo destino es el 1. Las listas deben estar implementadas como std::list o std::vector

4.3.3.2.3. Generalidad

Estos tipos de listas están definidos como templates, así como el grafo, luego podemos decidir el tipo de contenedores y de salida.

Todos los parámetros de la búsqueda pueden ser reimplementados como queramos.

Heurísticas: la heurística empleada ha sido la distancia euclídea por dar los mejores resultados sobre este tipo de grafos valorados.

Visitantes: los visitantes son funciones que se ejecutan sobre nodos, cuando sucede un tipo determinado de acción. Por ejemplo: astar_goal_visitor, es la función que define si el nodo es solución o no.

4.3.3.2.4. Funcionamiento

El algoritmo recorre el grafo de manera eficiente, modificando la heurística en cada estado alcanzado, y cuando encuentra el camino lo devuelve en forma de lista de puntos, con un valor asociado (coste) que define la distancia recorrida.

4.3.3.3. Búsqueda sobre Rejilla

4.3.3.3.1. Detalles

Entrada: Rejilla (tabla de celdas), Nodo Inicial, Nodo Final.

Salida: Camino Solución.

Descripción: Dado un mapa en forma de rejilla busca el camino entre una celda inicial (estado inicial) hasta una celda final (estado final).

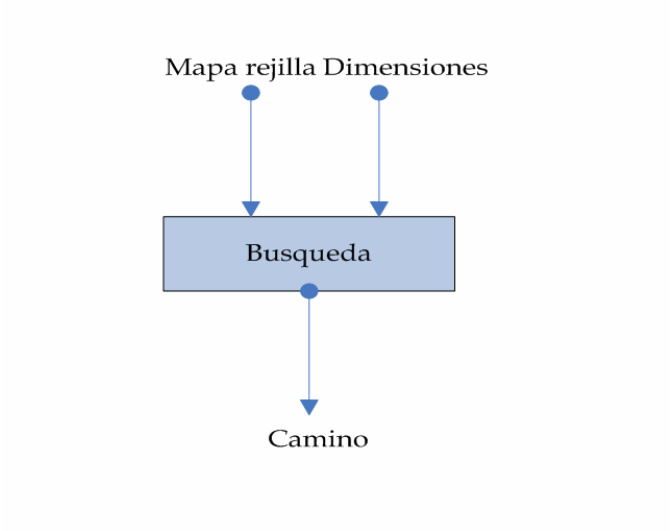


Fig 22. Entrada y salida de Busqueda

4.3.3.3.2. Requisitos

El nodo inicial y el final deben contener toda la información necesaria que se propagará a lo largo de la búsqueda:

- Restricciones de espacio: Un rectángulo definido por el punto arriba izquierda y el abajo derecha, define la zona de búsqueda sobre la rejilla.
- Valor por celda. El coste para pasar de una celda a otra
- Rejilla.
- Rango máximo de actuación. Se puede poner un límite donde a partir de ahí, los estados siempre son válidos.

Se deben definir antes de la búsqueda el nodo inicial y final.

4.3.3.3.3. Arquitectura y Diseño

EstadoEnrejado

El estado contiene toda la información necesaria y las funciones que permiten el despliegue dinámico de los estados sucesores válidos, es decir ,la construcción dinámica del grafo.

Se basa en la descomposición de celdas del espacio libre, este método se explicará más a fondo en el Planificador Local.

Estado de Búsqueda

El estado de búsqueda o Nodo del A* entonces se compondrá por:

- Los valores asociados a la heurística.: f(x) = g(x) + h(x)
- Un EstadoEnrejado

Tiempo Real

La búsqueda sobre rejilla es mucho más específica por la necesidad de que funcione en tiempo real. La construcción de un grafo sobre el espacio de estados es inviable, ya que para un tamaño de 1000x1000 existen aproximadamente 8 millones de posibilidades de conexión.

Así que hemos optado por la generación de los estados conforme va avanzando la búsqueda. Esto nos permite, gracias al A* una utilización mínima de los recursos de espacio y tiempo.

Montículos y Gestión de la memoria

No entraremos en conceptos de funcionamiento del A* por ser de sobra conocido.

Se usa una clase llamada FixedSizeAllocator, que es un manejador de memoria para objetos de un tipo especificado previamente por un template.

Con esto se consiguen inserciones y eliminaciones constantes o(1). Así para búsquedas pequeñas el algoritmo funciona en tiempo real. Para búsquedas grandes se necesita el uso de estructuras como árboles, cuadtrees u octrees.

La generación del grafo se hace de manera dinámica, se van expandiendo nodos según se avanza en la búsqueda. El único límite o restricción que presenta es el impuesto por el hecho de usar un manejador de memoria con un número limitado de nodos. Cuando se supera ese límite, o por alguna casualidad no se puede alcanzar el nodo destino, el algoritmo dice que no encuentra solución. Para ello hemos implementado que devuelva el camino aproximado al destino. Este nunca será óptimo, e incluso puede ser que solo posea un nodo, ya que es el camino con menor coste. Pero se puede utilizar si se necesita, para aproximaciones a obstáculos, salida de bloqueo, etc.

El principal problema de las búsquedas sobre rejilla es la posibilidad de la expansión de muchos nodos y por lo tanto la velocidad de tratamiento de las listas internas del A*. Así, aun reduciendo el mundo para solo la vista de los sensores el número de nodos abiertos para una distancia de 3 metros cuadrados, y con un factor de precisión de celda por centímetro, se pueden llegar a abrir 90000 nodos. Para manejar todo esto necesitamos una representación de las listas internas de la búsqueda, que tengan unos tiempos asequibles.

Para las listas de nodos abiertos y nodos cerrados se utiliza un **min-heap**, es decir, un montículo con orden de menor a mayor.

Operation	Binary	Binomial	Fibonacci		Pairing		Leftist
	worst-case	worst-case	amortized	worst-case	amortized	worst-case	worst-case
find-min	O(1)	O(log*n*)	O(1)	O(1)	O(1)	O(1)	O(1)
delete-min	O(log*n*)	O(log*n*)	O(log*n*)	O(*n*)	O(log*n*)	O(*n*)	O(log*n*)
insert	O(log*n*)	O(log*n*)	O(1)	O(1)	O(1) or O(log*n*)	O(1)	O(log*n*)
decrease-key	O(log*n*)	O(log*n*)	O(1)	O(1)	O(1) or O(log*n*)	O(1)	O(log*n*)
merge	O(*n*)	O(log*n*)	O(1)	O(1)	O(1) or O(log*n*)	O(1)	O(log*n*)

El uso y la aplicación de este método se verá en el Planificador Local.

4.3.3.3.4. Funcionamiento

Una de las curiosidades sobre nuestra búsqueda es que se realiza en pasos y por lo tanto puede ser cancelada.

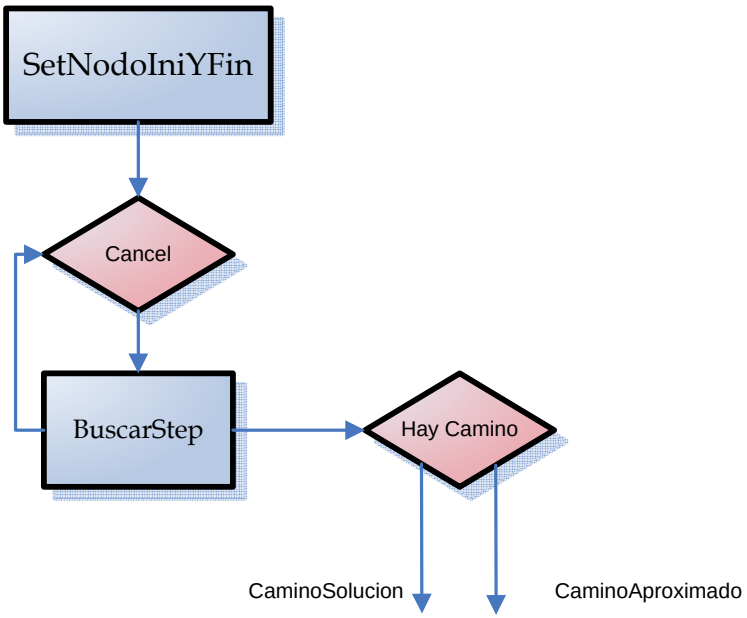


Fig 23. Funcionamiento de busqueda en enrejado

4.3.4. Planificador Global

4.3.4.1. Introducción

El planificador global es como si le explicáramos a un ciego por dónde tiene que ir sin avisarle de las incidencias. Dado un entrono conocido todos los cálculos se pueden hacer off-line. Luego no hace falta que se utilicen algoritmos de tiempo real. Aun así hemos decidido que sean suficientemente rápidos para la posibilidad de cálculo instantáneo.

Para ello usaremos un grafo de visibilidad como representación del entorno y una búsqueda sobre grafo de adyacencia, explicada anteriormente en el apartado *Búsqueda*.

Para ello hemos usado un modelo de robot circular. En nuestro caso tenemos suerte, porque nuestro robot físico es circular. Es más fácil navegar por la simetría del robot (el *espacio de configuraciones se reduce a 2D*).

4.3.4.2. Detalles

Entrada: Trayectoria, compuesta por un objetivo inicial y un objetivo final.

Salida: Trayectoria compuesta por un punto inicial, un punto final y puntos intermedios.

Descripción: Dado un mapa del entorno, calcula el camino libre de obstáculos entre dos puntos del mapa.

4.3.4.3. Grafos de visibilidad

Hemos decidido utilizar como método de planificación global para nuestro robot los grafos de visibilidad como hemos comentado anteriormente. Las decisiones que nos han llevado a elegir este método y no otros han sido:

- Mediante este método hemos podido construir un algoritmo de bajo coste computacional que resuelve perfectamente el problema de la planificación global.

- El entorno sobre el cuál realiza la búsqueda nuestro planificador global puede llegar a ser la totalidad del mundo sobre el que puede moverse el robot. Si utilizásemos otros métodos de planificación, como por ejemplo el enrejado de celdas, el número de nodos sobre los cuales deberíamos realizar la búsqueda sería inaceptablemente alto. Sin embargo con los grafos de visibilidad, dado que solo se consideran nodos del grafo de búsqueda a los vértices de los distintos obstáculos, al punto inicial del que parte el robot y al punto final al que debe llegar, conseguimos un espacio de estados de búsqueda más que razonable.

- Define segmentos rectilíneos de conexión entre los distintos puntos por donde debe pasar el robot, de esta manera se facilita bastante el trabajo de seguimiento de la trayectoria por parte del control de movimientos de bajo nivel.

El único problema teórico y práctico de utilizar grafos de visibilidad es que considera al robot como un punto en el plano, cuando en realidad no es así (el robot ocupa una superficie muy superior a un solo punto).

Solventaremos este problema (como se explica en detalle en el siguiente apartado) mediante la construcción de un entorno sobredimensionado a partir del original, sobre el cual se puede aplicar de la forma descrita la planificación basada en grafos de visibilidad.

4.3.4.4. Modelado circular del Robot Móvil

El algoritmo de planificación propuesto consta de dos fases fundamentales: i) una modificación inicial del entorno, para expandirlo según el radio ρ del círculo que modela el robot, y ii) una segunda fase de construcción del grafo de visibilidad para la búsqueda de una ruta óptima según una minimización de cierta función de coste. A partir de esta ruta, se obtiene una función ruta c(λ).

Esta consideración de robot circular para vehículos que pueden modelarse de manera geométrica por la mencionada forma, resulta especialmente eficiente en cuanto al método propuesto. Este hecho se debe a que se calcula una ruta en el mapa a un coste computacional bajo, por estar basado en cálculos geométricos.

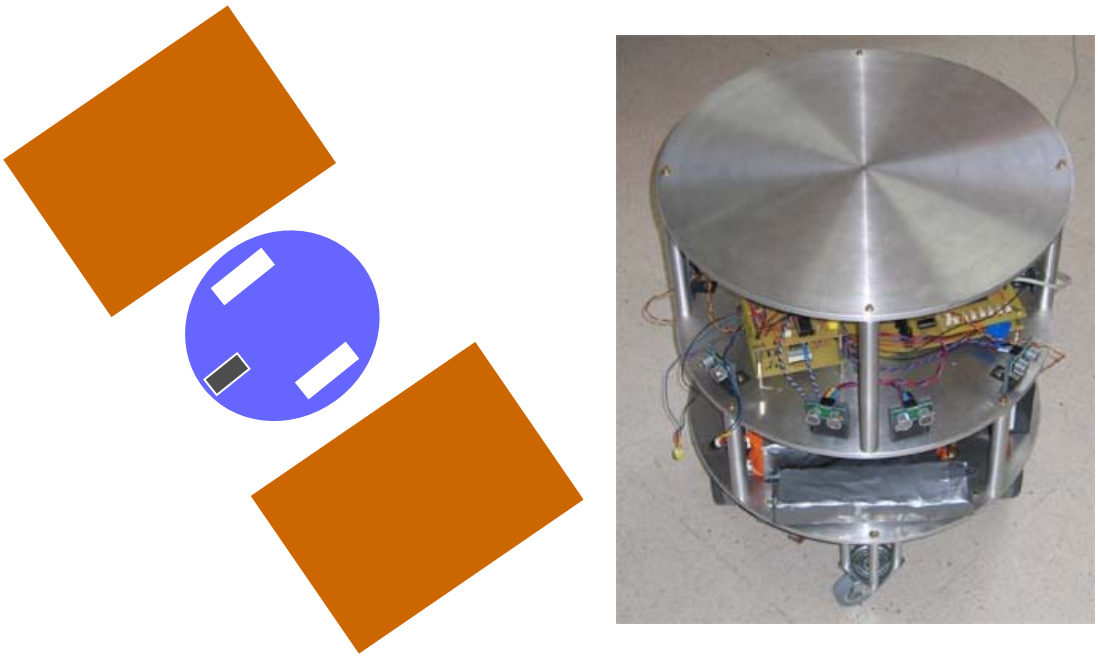


Fig 24. Robot circular izquierda, nuestro robot derecha

Propiedades

Nuestro robot móvil tiene 3 grados de libertad respecto a una referencia: posición en el plano (X,Y) y orientación (Q)

Idealmente, independientemente de donde inicie, el robot debe poder moverse a cualquier posición y orientación (X,Y,Q)

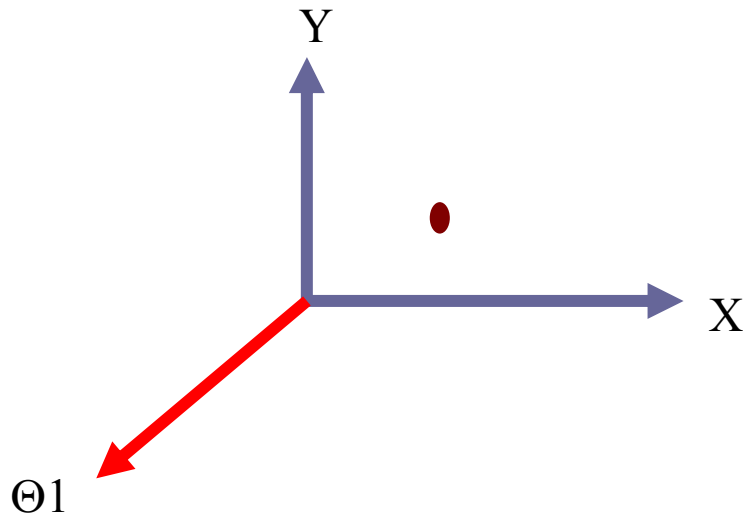


Fig 25. Representación del robot en el espacio cartesiano y orientación

Para simplificarlo aún más, el movimiento será definido por la velocidad de las ruedas, así el giro sobre si mismo es una rueda sola en movimiento. Dado esto, solo necesitamos planificar en un espacio en 2D.

4.3.4.5. Eliminación de la restricción de un robot puntual

Nuestra primera opción era la dilatación de los obstáculos, pero dada nuestra representación de los mismos, una lista de cuatro puntos, decidimos usar la técnica de construir un entorno sobredimensionado del mapa original.

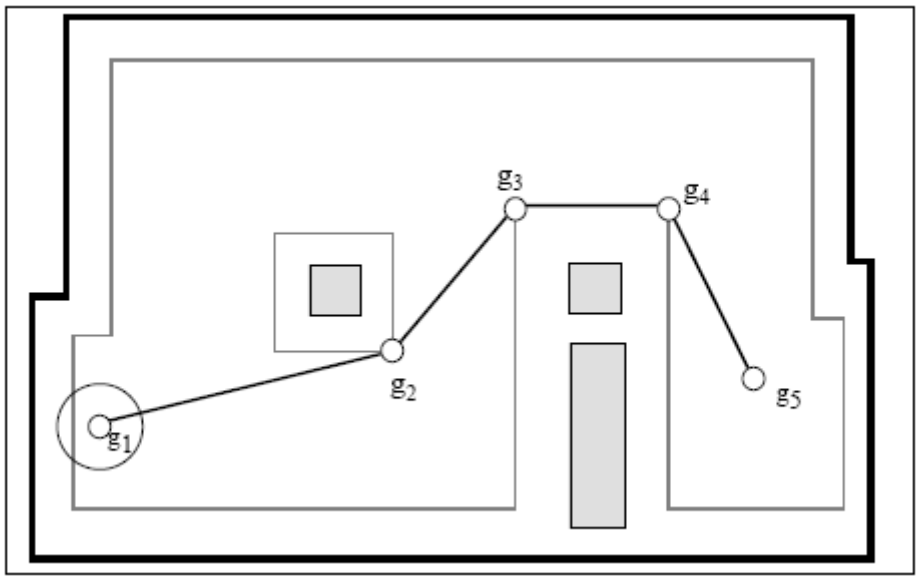


Fig 26. Mapa sobredimensionado de un entorno

La idea fundamental se muestra en la figura, donde la ruta definida por la secuencia ordenada de nodos $G=\{g1,g2,g3,g4,g5\}$, ha sido planificada mediante el uso de grafos de visibilidad y una versión sobredimensionada del modelo del entorno. La obtención de este nuevo modelo implica la utilización de un factor de expansión que garantice que la ruta planificada resulte segura. Usando un modelo de robot circular sobredimensionamos con un factor de expansión igual al radio de la circunferencia que lo modela.

Engrosado

La expansión de un obstáculo b_i conlleva añadir un conjunto de configuraciones adicionales $e(b_i)$ a $b_i(q)$, para construir la versión expandida del obstáculo *b2 (ref [2]):*

$$b2_i(q)=b_i(q)+e(b_i)$$

Donde $e(b_i)$ es una función que añade las configuraciones inmediatamente próximas a los límites del obstáculo. En el caso de un único obstáculo b_i que esté modelado mediante bp_i, su expansión significa la realización de una serie de modificaciones sobre el polígono que lo modela. Si el polígono es convexo, el algoritmo de expansión resulta fundamentalmente un escalado de sus aristas.

Debido a las acciones verificadas en el paso anterior, algunos de los elementos de E1 se han eliminado al fundirse con el entorno E1 para formar E2. Sin embargo, existe la posibilidad de solapamiento de

configuraciones por parte de algunos de los obstáculos expandidos del conjunto *B2 (engrosados)*. Según el planteamiento del espacio de configuraciones, este hecho carece de importancia, ya que lo esencial para efectuar la planificación consiste en el cálculo del conjunto de configuraciones libres.

Distancia de Seguridad

No solo basta con encontrar un camino libre de obstáculos a partir del conjunto de configuraciones libres, sino encontrar el que nos asegure que no haya ninguna posibilidad de colisión. Además queremos conseguir que sin usar diagramas de vornoi, el camino este lo más separado de los obstáculos posible. Dado que el uso del robot se va a centrar en pasillos, el factor de expansión tendrá que ser más grande que el radio del robot.

Si tenemos la posición del robot, según se va moviendo por el entorno el error acumulado de la posición se va incrementando ostensiblemente. Por estas razones introducimos una distancia de seguridad.

$$b2_i(q)=b_i(q)+e(b_i)$$

Donde e(b_i) = radio_robot + distancia_seguridad.

4.3.4.6. Bloqueo y Replanificación

En muchas situaciones el robot queda bloqueado en el espacio de configuraciones creado a partir de la sobredimensión o engrosado de los obstáculos del mapa original. Pero eso no quiere decir que el robot no pueda pasar.

Si tenemos un conjunto de obstáculos sobredimensionados *Ci*, donde no existe ruta c(λ). Buscaremos un conjunto *Ci+1* que nos permita el paso. Para ello reduciremos la distancia de seguridad y replanificaremos.

distancia_mínima + radio_robot = ***emin(bi)***

Reduciremos un 20% o la petición del módulo, la distancia hasta alcanzar *emin(bi)* y si no se encuentra ruta ci+1(λ), es que no existe camino posible alguno.

Para replanificar recalculamos la trayectoria con el espacio de configuraciones nuevo.

Existe un límite de replanificaciones que esta definido como parámetro del módulo, es decir, que se puede modificar si recompilar nada.

Trayectorias Replanificadas por la entrada

Se pueden recibir trayectorias de replanificación por el puerto de entrada, que se tratarán de manera diferente en el procesamiento de la trayectoria.

4.3.4.7. Funcionamiento

Para explicar el funcionamiento del Planificador basta con ver el diagrama de flujo de la función ProcesarTrayectoria(Trayectoria *t). Se trata de un diagrama recursivo, ProcesarTrayectoria se llama a sí mismo.

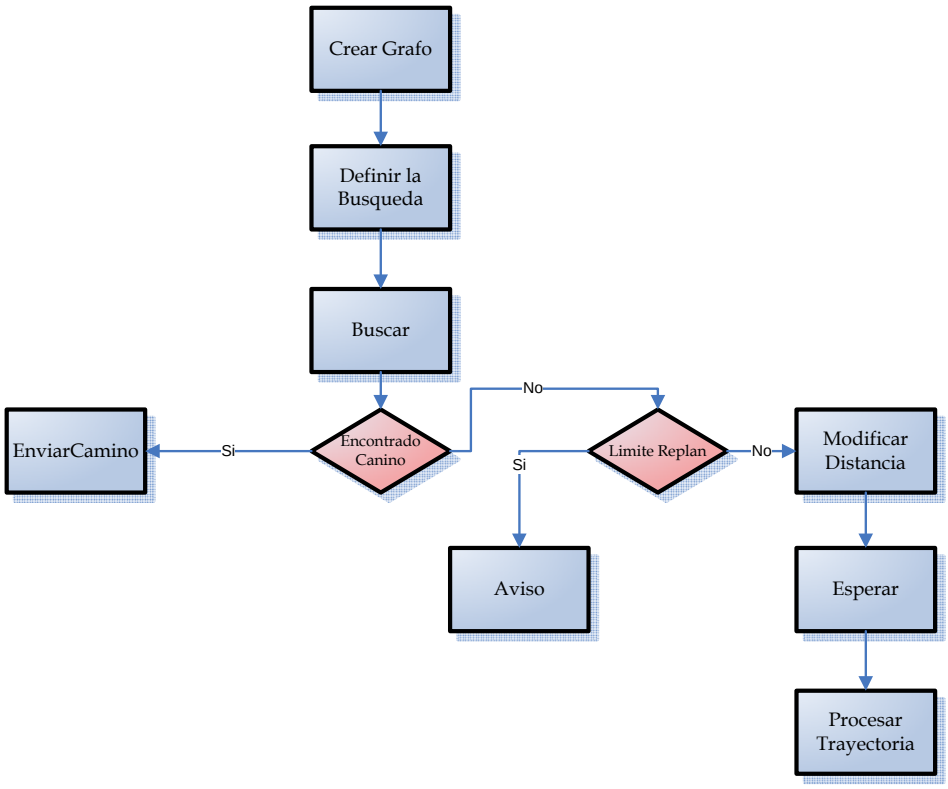


*Fig 27. Diagrama de flujo de la función ProcesarTrayectoria(Trayectoria *t)*

4.3.5. Planificador Local

4.3.5.1. Introducción

El planificador local, son los ojos del robot cuando se desenvuelve solo por el mundo. Solo se centra en el espacio que rodea al robot y que define el alcance de los sensores. Es un módulo crucial para la navegación libre de colisiones.

La idea de este módulo es que aunque no tuviera planificación global, pudiera encontrar rutas hasta un objetivo cualquiera sin colisionar. Para ello representamos el mundo como una rejilla donde quedan descritas las zonas con obstáculos. Por medio de una búsqueda A* podemos trazar caminos desde un punto a otro del mapa local.

Este módulo tiene que funcionar en tiempo real, ya que de eso depende la eficiencia del robot y la aparente inteligencia en los movimientos. Tenemos las ventajas de que el mapa local es restringido y la búsqueda muy rápida, también hay que tener en cuenta que no siempre se modifica el camino, simplemente se comprueba si existe peligro de colisión en la trayectoria que se esta siguiendo y si no es así no hay que poner en marcha la búsqueda sino simples cálculos aritméticos.

4.3.5.2. Detalles

Entrada: Trayectoria compuesta por una lista de puntos en el espacio con un origen y un destino. La posición y orientación del robot .

Salida: Lista de Puntos Consigna separados x centímetros. Velocidad máxima permitida para ese tramo.

Descripción: Módulo que dado una trayectoria a seguir controla de forma local que esa ruta este libre de obstáculos sirviéndose de los sensores y caso de encontrar un obstáculo en el camino esquivarlo, creando un camino alternativo.

4.3.5.3. Enrejado

Para la representación del mapa hemos optado por un enrejado, un modelo de descomposición en celdas (ya explicado anteriormente en la introducción) en el cual se divide el espacio de trabajo del robot en una cuadrícula. La conexión entre las celdas es 8-Conectado.

La implementación a bajo nivel es un array de enteros sin signo que toman los valores de: LIBRE, OBSTACULO, SEGURIDAD, OBSTACULO_LOCAL, SEGURIDAD_LOCAL.

Podemos saber si una casilla esta libre si es igual a cero.

Normalmente solo se necesitan para la planificación global OBSTACULO, SEGURIDAD, y para la planificación local OBSTACULO_LOCAL, SEGURIDAD_LOCAL. Pero se pueden utilizar todos los valores para diferentes modos de comportamiento.

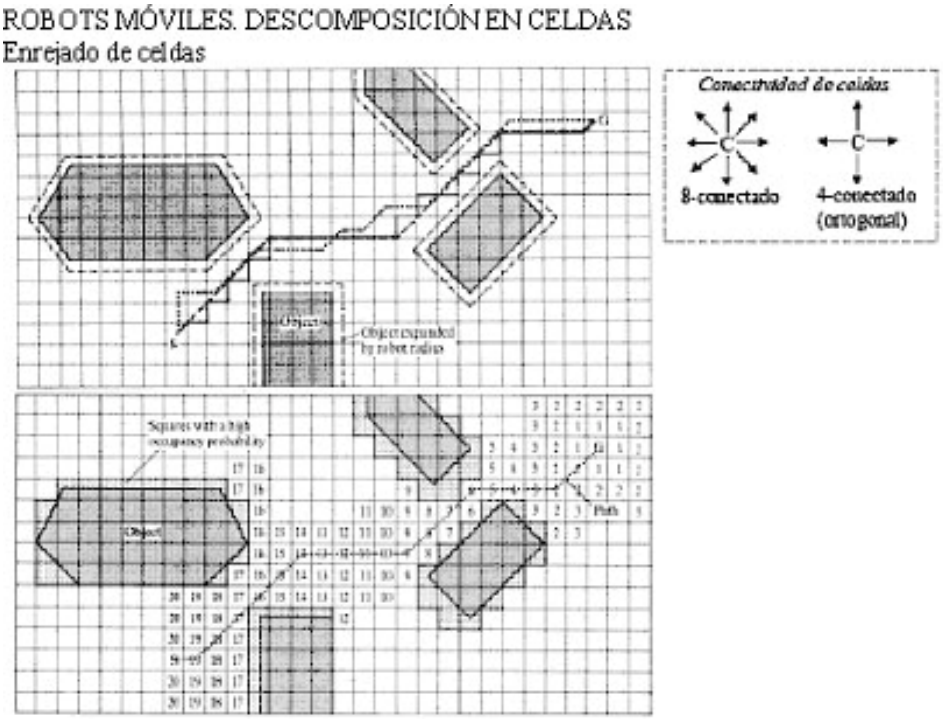


Fig 28. Ejemplo de enrejado

La estructura de memoria requerida para anticipar y optimizar el cálculo de trayectorias ante un obstáculo y para clasificar su dinamismo se ha implementado mediante un mapa local de rejilla de 20x20 metros dividido en celdas de 20x20(cm.) que se mueve solidario con el robot. En este caso se trata de elaborar a partir del mapa local de rejilla el estímulo

obstáculo desencadenante del comportamiento Esquivar_Obstáculo. En función de las relaciones espacio temporales de los distintos obstáculos que pueden estar presentes en la escena se selecciona la estrategia de navegación más adecuada. El estímulo obstáculo se define como un grupo de celdas con estado ocupado conectadas entre sí y pertenecientes al Área_de_Movimiento próxima al robot. En esta aproximación cada obstáculo se caracteriza por: posición y tamaño.

El Área_de_Movimiento se define como el área semicircular centrada en la posición del robot, (*xr, yr, ϕr*) y de radio *rr* = 19cm, al ser el área relevante en la colisión con obstáculos.

$$\begin{aligned} Area\ Movimiento = & celdas\ (x_{c,i}, y_{c,i}) \in Mapa\ Local/ \\ & (x_{c,i} - x_r)^2 + (y_{c,i} - y_r)^2 < r_m^2 \text{ y} \\ & \arctan(\frac{y_{c,i} - y_r}{x_{c,i} - y_r}) \in (\theta_r - (\pi/2), \theta_r + (\pi/2)) \end{aligned}$$

En nuestro caso al poder realizar movimientos de marcha atrás vamos a relajar las restricciones para que tanto la media luna frontal como la trasera, forme parte del Área de Movimiento.

Posición del obstáculo

Las coordenadas de un obstáculo corresponden al valor medio de las coordenadas de los puntos centrales de cada una de las celdas que lo forman.

$$x_i = \frac{c_{1,x} + c_{2,x} + ... + c_{n,x}}{n}\ ,\ y_i = \frac{c_{1,y} + c_{2,y} + ... + c_{n,y}}{n}$$

Sin embargo pensando en la función de este estímulo en los comportamientos de navegación segura, es más relevante disponer de un valor aproximado del centro del mismo, que perseguir su posición exacta mediante algoritmos geométricos más complejos.

Algoritmo de segmentación

El primer paso en la construcción del estímulo obstáculo a partir del mapa local es la segmentación, es decir, el cálculo desde la rejilla de ocupación de las celdas que conforman cada obstáculo. Se trata de un algoritmo de barrido de imagen en un único paso, que procede en la siguiente forma:

1. Construcción de la lista de celdas obstáculo. Ésta es una lista que almacena los índices de las celdas del mapa que pertenecen al Área_de_Movimiento y tienen estado ocupado.

2. Agrupamiento de las celdas de la lista de celdas obstáculo. Para ello se recorre la lista anterior ordenadamente. Se asigna la celda 1 al obstáculo 1 y se buscan todas las que están conectadas con ella, asignándolas al obstáculo 1. Este proceso se realiza de forma recursiva con todas y cada una de las celdas incorporadas hasta que no haya más celdas conectadas o se llegue al final de la lista de celdas obstáculo. Para evitar la repetición del proceso sobre celdas ya contadas, se mantiene otra lista, la lista de celdas contadas que incluye las celdas ya asignadas a un obstáculo. Finalizado el proceso sobre el obstáculo 1, se continúa con las celdas de la lista de celdas obstáculo que no pertenecen a la lista de celdas contadas.

Al final de este paso, dada la rejilla de ocupación en el instante t, mapa(t), se obtiene una colección de objetos Oi(t). Cada uno de ellos definido por las celdillas cOi(t)(n) que lo conforman, sin caracterizar aún su tamaño.

Corrección del Algoritmo de segmentación

Puesto que los módulos de bajo nivel ya nos dan un obstáculo con anchura y profundidad en el espacio, es decir hay un módulo intermedio entre los sensores y el planificador local, se simplifica la tarea. Y no se necesitan tantas listas. Con una lista de obstáculos locales, y un barrido de la imagen, tenemos la rejilla calculada.

4.3.5.4. Optimalidad

La necesidad del camino perfecto en esta planificación no existe, ya que buscamos una navegación libre de colisiones y no nos importa el camino elegido. Aún así siempre se intenta el camino más corto.

Si el robot esta siguiendo una trayectoria *Ti* y existe un obstáculo en ella, el planificador local buscará una trayectoria alternativa *Tl* libre de colisiones. Esta trayectoria no es necesariamente óptima:

Aproximación e información incompleta: Se puede evitar un obstáculo de forma aproximada, porque todavía no se conoce bien el espacio de configuraciones local, ya que falta mucho para llegar al obstáculo y los sensores no han recogido toda la información. Tal como se ve en la figura siguiente.

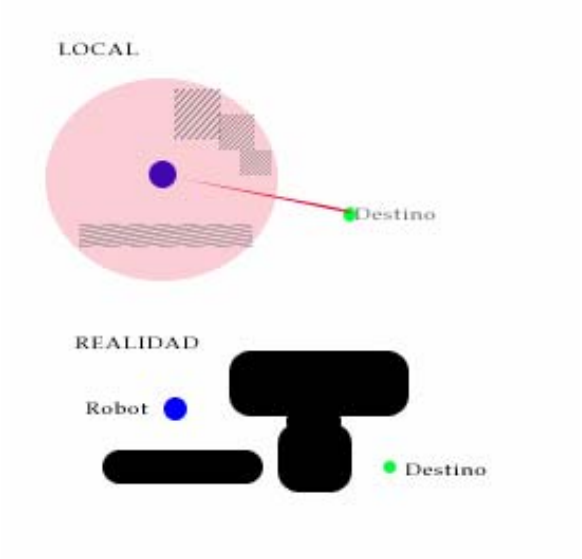


Fig 29. Diferencia entre realidad y percepción del robot

Ruido: Se pueden producir ruidos captados por los sensores que engañen al planificador local, encontrando trayectorias que no se corresponden con la realidad.

Sobredimensionado de los obstáculos: Dada un factor *fe* de sobredimensionado de los obstáculos sobre el enrejado conseguimos un conjunto de espacios de configuraciones modificado *Ci*. Una ruta c(λ) óptima sobre el espacio real de configuraciones puede no ser válida sobre *Ci* por la modificación de los obstáculos. Es decir la distancia de seguridad provoca la no optimalidad del camino.

Comportamiento Inteligente (humano): No por elegir una trayectoria más corta pareces más inteligente, sino por elegir la trayectoria con menos posibilidades de chocar con alguien o darte con algún objeto. Cuanto más información posea el planificador, más inteligente parecerá el robot. Actualmente poseemos sensores y un mapa, con lo que la información es muy alta si no existen obstáculos en movimiento. En este caso, como no sabemos donde se va a mover el objeto, ya que ha aparecido de repente, la solución siempre es parar y analizar el entorno después de un determinado tiempo transcurrido.

Heurísticas: Diagonal y Distancia Euclidea.

4.3.5.5. Regiones de Peligro e Inserción de obstáculos

La región de peligro al entrar en funcionamiento el evento obstáculo, es la ya mencionada área de movimiento.

Un obstáculo será insertado en el mapa sí y solo sí pertenece a la zona de sensores definida. Solo se replanifica en caso de que el obstáculo este en zona de trayectoria. Si un obstáculo aparece en zona del robot, este se detiene.

Todos los objetos localizados en el área de sensores son insertados porque el robot puede moverse en sentido contrario si así lo manda la ruta y por lo tanto cambiar de sentido, incluso marcha atrás.

4.3.5.6. Funcionamiento

El funcionamiento del planificador local es bastante intuitivo pero detrás encierra una gran complejidad, por la gran cantidad de posibles casos. Simplificando el algoritmo de navegación seguido por el local simplemente comprueba si se ha llegado al final, y si no si se procesa una trayectoria nueva o se sigue con la misma trayectoria, dependiendo de la posición del robot o si esta bloqueado.

Podemos distinguir entre dos tipos de puntos:

Puntos de Consigna Locales: Puntos que han sido planificados por el local y que definen la trayectoria final.

Puntos de Consigna Globales: Puntos planificados por cualquier tipo de planificador global o módulo que conoce toda la información acerca el mapa donde se desplaza el robot.

El objetivo final siempre es el *destino* y el inicial se va modificando conforme se va avanzando en la ruta $c(\lambda)$, o por modificación expresa al esquivar un obstáculo.

4.3.5.6.1. Algoritmo de navegación local

Si Comienzo || ObjetivoFinalAlcanzado →

Si HayTareasPendientes → TrayectoriaActual = PriemeraTarea

Sino → TrayectoriaActual = LeerTrayectoriaPuerto

Inicialización de variables

PorcesarTrayectoria(TrayectoriaActual)

Sino →

// Objetivo Intermedio

Si ObjetivoGlobalAlcanzado →

Inicialización de Variables

ProcesarTrayectoria

Sino → CorregirDesviacion // Si hace falta

Si Bloqueado || RobotEnSolapado → ProcesarTrayectoria

Para entender el funcionamiento interno y comprender mejor el algoritmo de navegación local que se emplea en este módulo hay que centrarse en la función ProcesarTrayectoria. Dentro del algoritmo se utilizan 4 conceptos importantes:

- Calcular Zona.
- Dividir Trayectoria.
- Corregir Camino.
- Esquivar.

Algoritmo de navegación local interno

Si Trayectoria nueva →

Si HayPuntosConsignaGlobalesIntermedios→Fin = Siguiente

Sino → Fin = ObjetivoFinal

CalcularZona

Si DividirTrayectoria →

// No hay Obstáculo en el camino

Si ¬ CorregirCamino → EnviarPuntosConsigna

Sino → EsquivarObstáculo

Elige un nuevo punto de destino e intenta dividir la trayectoria en puntos separados *x* centímetros, comprobando en cada uno si están ocupados por algún obstáculo.

Si no hay obstáculos en la trayectoria envía los puntos consigna calculados.

Antes de enviar los puntos comprueba si se ha desviado mucho de la trayectoria original marcada por el planificador global, ya que cuando esta controlando el planificador local no tiene información sobre todo el entorno y por lo tanto es peligroso.

Si en alguno de los puntos ha visto que existe un probable obstáculo, es decir en nuestra representación de enrejado, entonces DividirTrayectoria devolverá *false* y entonces pasará a esquivar.

Calcular Zona

Calcula la región local es decir la circunferencia que tiene como radio el alcance de los sensores. En particular los de ultrasonidos, que en nuestro robot son los que tienen más alcance.

Dividir Trayectoria

Se divide la recta que une el punto inicial y el final, en puntos separados *x* centímetros , pertenecientes a la recta.

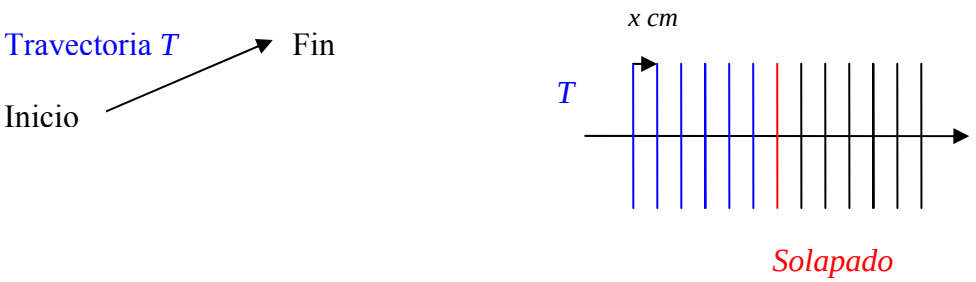


Fig 30. División de trayectoria

Comprobación de punto ocupado

Dado x,y ε Mundo Real R, y dado Cr (conjunto de configuraciones de la rejilla)

Discretización a enrejado:

Xc = X / factor de reducción(x)

Yc = Y / factor de reducción(y)

Existirá Obstáculo en la trayectoria ⇔ Existe(x,y) ε Trayectoria y (Xc,Yc) = OBSTÁCULO

Punto En Alcance: Todo punto que pertenece a la zona definida por el alcance de los sensores (Área de movimiento).

Punto Solapado: Todo punto que se recalculará posteriormente, para que parezca que el robot tiene más predicción.

Punto Válido: Punto que está en alcance, no es punto ocupado ni solapado.

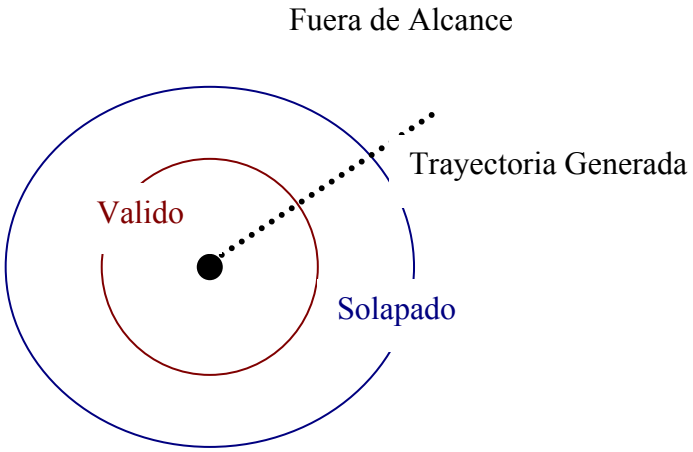


Fig 31. Áreas del planificador local

Solapado

El solapado es una serie de puntos que se recalculan para dotar de más predictibilidad al robot. Así hay una zona que se ha comprobado anteriormente si hay obstáculos, pero que el robot no llegará hasta la siguiente división de trayectoria, y por lo tanto se comprobarán de nuevo esos puntos. La sensación que transmite a las personas es que el robot ve hasta mucho más lejos y reacciona anteponiéndose al suceso.

Esquivar Obstáculos

Para esquivar un obstáculo buscamos nuevos puntos de referencia que eviten el obstáculo si hay riesgo de colisión, formando un camino alternativo con el inicio y el fin pertenecientes al camino calculado globalmente.

Par encontrar esos nuevos puntos de consigna ampliamos la zona de los sensores, construyendo la zona de búsqueda. Sabemos que:

Si x,y pertenecen a la zona de búsqueda pero no a la zona de sensores, pueden ocurrir dos casos:

a) Existe obstáculo Global: No hay problema porque esta contemplado en el Global y en una replanificación global se encontrará camino.

b) No hay obstáculo Global: Nunca puede aparecer un obstáculo fuera de la zona de los sensores. Ya que como hemos explicado antes, aunque los sensores lo detecten, nunca se insertará en el mapa.

Así garantizamos que el punto objetivo este libre de obstáculo.

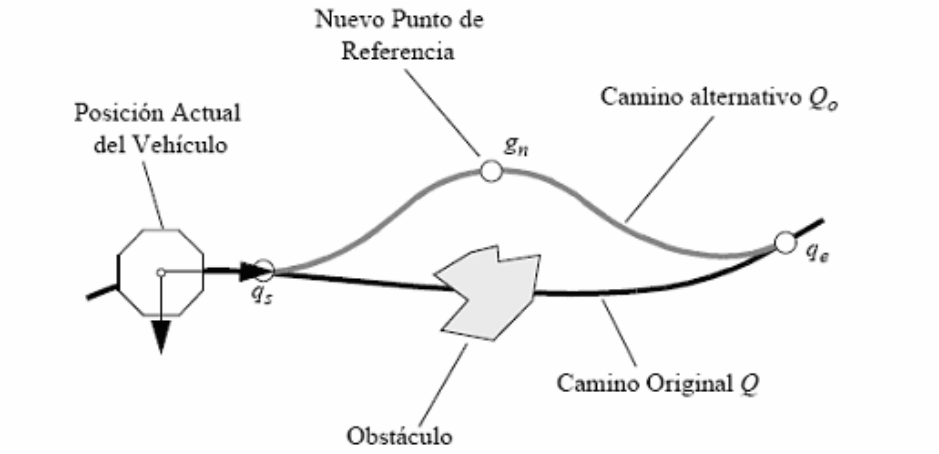


Fig 32. Búsqueda de camino alternativo al encontrar obstáculo

El algoritmo de navegación para esquivar el obstáculo es el siguiente:

Si Objetivo dentro del área de los sensores →

Fin = Objetivo

Sino →

Fin = Calculamos el punto perteneciente a la recta que une la posición del robot y el objetivo y que corta con el radio de búsqueda *r*.

Si BuscarCamino(Posición Robot, Fin) →

Mandamos los puntos Consigna nuevos.

Sino →

// No se ha encontrado camino

Si número de Replanificaciones > Máximo →

ReplanificaciónGlobal

Sino →

Modificar Distancia Seguridad Local

Entramos en Bloqueo

Cálculo del punto de corte

Dada la recta entre dos puntos: (y-y0) = m (x –x0) donde m = (y1 – y0) / (x1 –x0).

La circunferencia: Pi r^2 donde r = radio de búsqueda.

Tomando (x1,y1) = fin y (x0,y0) = Inicio, obtenemos el punto de corte entre las dos ecuaciones, que será el punto destino para la búsqueda A*.

Replanificación Local y Global: Bloqueo

Cuando entramos en bloqueo, se intenta una replanificación local del camino, con una distancia de seguridad reducida. Si por alguna casualidad el problema no es local, sino global pues hay que avisar al planificador global, para que replanifique la trayectoria con los nuevos obstáculos.

La implantación de esta comunicación se hace realimentando la entrada del planificador global con la trayectoria pedida por el local.

5. Módulo de Mapa

5.1.Introducción

Hemos hablado sobre diferentes representaciones para el entorno donde se desenvuelve el robot y cada una totalmente diferente, pero al fin y al cabo todas se resumen en la información sobre el mapa que define este entorno. Entonces, por qué no centralizar todo y crear un modelo de servicio. El módulo contiene toda la información pertinente relacionada con el mapa y por medio de peticiones los demás módulos pueden reclamar una modificación en el mapa o una representación del mismo.

Además hemos elegido una representación del mapa como XML, así que este módulo se encargará de leer y construir el mapa a partir de un fichero XML y los parámetros elegidos para su creación.

Conseguimos uniformidad y control central, y si en algún momento se decide que no es la mejor forma de representación se puede sustituir por otro, sin modificar los módulos restantes como el planificador global y local.

5.2.Detalles del módulo

Entrada: Peticiones y Obstáculos

Salida: Mapas en sus diversas representaciones.

Descripción: Módulo encargado de centralizar todos los datos relacionados con el mapa así como las funciones asociadas.

5.3.Arquitectura y funcionamiento del módulo

5.3.1. Lectura del Mapa

Se lee un xml compuesto por dos tipos de datos:

- Obstáculo: es un paralelogramo definido por cuatro puntos
- Obstáculo Objetivo: Es un obstáculo que además posee un id

En esa lectura se recogen también:

- Alto del mapa
- Ancho del mapa
- Pixels por centímetro
- Centímetros por celda que queremos.

La lectura permite los parámetros de engrosado, seguridad y tamaño de la pared.

La pared es muy útil dado que muchas veces los sensores localizan objetos fuera del mapa así que se define en el xml un mapa más grande y se le pone una pared de *x* centímetros.

A partir de los tipos de datos se generan las diferentes representaciones del entorno: listas de obstáculos, lista de obstáculos engrosados, rejilla local, etc.

Lectura de objetivos

La lectura de objetivos se realiza en tiempo de ejecución así permite la modificación en ejecución de los objetivos y la carga cada vez que el robot se pone en modo guía.

Se hace mediante la lectura de un fichero xml que contiene:

- Id del objetivo
- Posición en el mundo real
- Sonido asociado al objetivo
- Tipo de objetivo

5.3.2. Peticiones

El módulo funciona como una aplicación de servicios. Se le hace una petición y este responde a ella mandando información a los puertos de salida. Las peticiones utilizadas hasta ahora son:

- Objetivos Extendidos: Ruta definida a seguir en modo guía
- Obstáculos: Lista de obstáculos.
- Distancia de seguridad global. Se puede pedir la reducción, el aumento o reestablecimiento del valor por defecto. Las posibilidades para incrementar o decrementar la distancia de seguridad son:
    - 10%
    - 20%
    - 50%
    - 75%
- Distancia de seguridad local. La petición tiene las mismas posibilidades que la de la distancia global.

Capítulo 5

Otros módulos y aplicaciones del sistema

1. Introducción

En este capítulo presentamos una visión detallada de los módulos restantes del sistema que no forman parte de su núcleo. Se trata de los módulos que hacen posible la interacción entre nosotros y el robot y también la interacción de nuestro guía autónomo con los visitantes del museo.

Se puede decir que este subsistema representa el "oído" y la "boca" de nuestro robot, sin los cuales a nosotros nos sería imposible entender su comportamiento y a él le sería imposible saber qué comportamiento esperamos de él.

2. Módulo de Simulación

2.1.Introducción

Cuando se realizan aplicaciones que dependen de otras se necesita una simulación para comprobar que todo está en orden y que el comportamiento es el esperado. Dado que en nuestro proyecto tenemos tres partes bien diferenciadas Alto-Nivel, Bajo-nivel y Robot físico, se han realizado varios módulos que pretenden emular el funcionamiento del robot.

2.2.SimuRobot

Este módulo ha sido implementado para poder probar la parte de alto-nivel, sin necesidad del robot físico y el software de bajo-nivel.

2.2.1. Detalles

Entrada: Trayectoria en forma de puntos de consigna, Velocidad.

Salida: Posición y Orientación.

Descripción: Simulación del robot de bajo nivel

2.2.2. Modelado

El modelado del robot es sencillo y sin ecuaciones físicas. Simplemente tiene las mismas entradas y salidas que tendría la parte de bajo-nivel conectadas de la misma manera.

El movimiento de este es ideal, cada tiempo *t*, cambia la posición al siguiente punto que ha recibido por la entrada una posición. Es decir que siempre llega al destino sin errores, ni problemas. Cada 100 milisegundos envía la posición a la salida.

Con este modelado podemos también comprobar si se han mandado puntos redundantes o demasiados pocos.

2.3.SimuEntorno

Este módulo ha sido diseñado e implementado para la prueba del comportamiento del robot de alto-nivel ante la aparición de obstáculos.

2.3.1. Detalles

Entrada: Posición del robot.

Salida: Obstáculos

Detalles: Simulación de los sensores.

2.3.2. Modelado

El modelado real de los sensores es arduo y largo, y necesita de mucho tiempo para el ajuste de parámetros, así que optamos por un modelo muy simplificado en el cual dependiendo de la posición del robot y de un factor de probabilidad, pueden aparecer obstáculos alrededor del robot.

Por medio de dos parámetros modificamos la velocidad de aparición de obstáculos y la probabilidad de que estén cerca del robot.

La posición del obstáculo es aleatoria y pese a que normalmente los sensores funcionan de manera predecible, con este método podemos averiguar que pasaría en situaciones imprevisibles, y comprobar el buen comportamiento del robot. Por ejemplo la detección de un obstáculo dentro del robot, o el bloqueo por multitud de obstáculos alrededor del robot.

3. Módulo de Entorno 2D

3.1.Introducción

El entorno virtual 2D constituye un interface visual con el usuario que puede observar la simulación de las evoluciones de un robot que se desplaza por el mapa siguiendo las trayectorias que hace el robot. Con esta aplicación, es posible evaluar y testar las funcionalidades que se han implementado sin tener que llegar a la integración del sistema en una entidad robótica real. Así como la monitorización del robot real en funcionamiento.

3.2.Detalles del módulo

Entrada: Posición, orientación trayectorias, mapa

Salida: El módulo no tiene salidas, ya que solo recoge la información muestra en la pantalla el movimiento del robot virtual.

Descripción: Módulo que permite el seguimiento del robot en el mapa

3.3.Arquitectura y funcionamiento del módulo

El módulo de entorno 2D (SalidaImagen) se conecta a todas las salidas de otros módulos de los que requiera información: posición, orientación, mapa, trayectorias globales y locales, puntos de consigna, etc.

A partir de esos datos lo va almacenando en una clase que se encargará de la parte gráfica, en forma de puntos y líneas. Cada cierto tiempo se llama al método pintar de esa clase, repintando y creando la sensación del robot en movimiento.

3.4.Implementación

Mediante funciones de inserción, variables que almacenan puntos y líneas, y las componentes siguientes podemos pintar en la pantalla sin problemas, gracias a la librería GTKmm (*ref [18]*)

DrawingArea: Área de dibujo con la que creamos una zona de pantalla drawable.

PixMap: mapa de pixels que nos permite dibujar sin pasarlo a la pantalla, y luego renderizárlo sobre esta cada vez que se refresque.

PixBuffer: Bufer de pixels que nos permite a partir de una imagen renderizarla sobre el mapa de pixels.

Image: Imagen proporcionada por el LectorMapa que se va modificando a cada instante.

4. Módulo de sonido

4.1.Introducción

Este módulo tiene como objetivo dotar al robot de la capacidad de reproducir sonidos para la explicación de las distintas piezas del museo y para interactuar con los visitantes. Recibe peticiones de reproducción de sonido por parte del módulo cerebro y se encarga de servir estas peticiones. Además informa al cerebro del tiempo de reproducción del archivo correspondiente para que éste pueda tomar las decisiones adecuadas sobre tiempos de paradas.

4.2.Detalles del módulo

Entrada: Este módulo recibe como entrada un identificador de un sonido a reproducir por parte del módulo cerebro.

Salida: Genera como salida la duración del archivo de sonido. Reproduce íntegramente dicho archivo.

Descripción: Módulo para la reproducción de ficheros de sonido en diversos formatos y que hará la función de "boca" del robot.

4.3.Arquitectura y funcionamiento del módulo

El módulo sigue las interfaces de comunicación con el pipeline, de modo que crea los búferes en la función de iniciar, a la vez que, hace uso de un fichero xml de configuración para obtener las rutas de los ficheros de sonido que corresponden al identificador de sonido que recibe como entrada.

Utiliza una cola para almacenar las peticiones de reproducción que le llegan otorgándoles igual prioridad, esto evita conflictos si se da el caso de recibir una petición de sonido cuando aún no se ha terminado de reproducir el archivo de otra petición anterior.

4.4.Bibliotecas

Para la interactuación de nuestra aplicación con el sistema de sonido instalado en el equipo que controla el robot hemos hecho uso de la siguiente biblioteca:

- fmod: biblioteca muy utilizada y que nos proporciona una API sencilla para reproducir sonidos en diversos formatos.

5. Sistema de comunicación con consola remota

5.1.Introducción

Este sistema tiene como objetivo implementar la comunicación entre el robot y una consola remota desde la que se pueda monitorear al robot y enviarle órdenes. Para ello hemos adoptado un esquema cliente-servidor, en el cual el robot actuará como servidor y la consola remota como cliente. Gracias al esquema de cliente-servidor obtenemos transparencia en cuanto al tipo de máquina y sistema operativo. Los procesos cliente y servidor se entienden con independencia de cómo sea la máquina y el tipo de sistema operativo que tenga cargado.

La comunicación se lleva a cabo mediante el protocolo de red TCP y, gracias a los dispositivos Wifi, esta comunicación puede llevarse a cabo sin la necesidad de cables, dotando al robot de una gran independencia.

5.2.Detalles

El sistema de comunicación está formado por tres módulos que realizan cada uno un tipo de operación. Esto se ha hecho así para dotar al robot de un sistema lo más modular posible. Así, si en un futuro se decide cambiar las órdenes a recibir, o el tipo de conexión de red utilizada, únicamente deberemos cambiar el módulo que se encargue de ello, sin tener que modificar los dos restantes. Los módulos por los que está formado el sistema de comunicación son *ServidorWifi*, *enviaDatos* e *interpreteWifi*. Los cuales explicaremos a continuación.

5.3.Submódulo servidorWifi

Este módulo es el encargado de realizar la transmisión de los datos mediante la comunicación de red (en nuestro caso via Wifi). Para la implementación de dicha comunicación hemos utilizado una librería de sockets, *C++ Socket Class for Windows (http://www.adp-gmbh.ch/win/misc/sockets.html)*, encontrada en internet.

La comunicación de red se lleva a cabo mediante el protocolo TCP (protocolo orientado a conexión) y mediante sockets SOCK_STREAM. Este tipo de sockets permiten comunicaciones bajo protocolos orientados a conexión (TCP) y tienen las siguientes ventajas frente a socketes SOCK_DGRAM que utilizan protocolo UDP:

- Fiabilidad de la transmisión: Ningún dato se pierde.
- Conservación del orden de los datos: Los datos llegan en el mismo orden en que fueron emitidos.
- No duplicación de datos. Sólo llega al destino un ejemplar de cada dato emitido.
- Se establece una conexión entre los dos puntos antes de los envíos, Después, en los datos que se envíen, no hay que especificar direcciones.

En resumen, utilizamos SOCK_STREAM pues lo que nos importa es la fiabilidad, aún a costa de una mayor complicación en la programación.

El módulo tiene como argumentos *num_conexiones* y *puerto* que refieren el número de clientes a los que se puede conectar (en nuestro caso usaremos 1, una sola consola remota) y el puerto al que se desea asociar el socket (que tendrá que ser el mismo en la consola remota) respectivamente.

El módulo *servidorWifi* consta de dos hilos de ejecución, uno de recepción y otro de envío.

El hilo de envío lee del puerto de entrada, que está conectado con el módulo *enviaDatos* un mensaje. Lo convierte a formato enviable y lo envía al cliente (consola remota).

El hilo de recepción de datos se encarga de recibir un mensaje via red (wifi en nuestro caso). Una vez recibido, lo transforma a formato *MensajeType*, que consta de una cabecera con el tipo de mensaje que es (tipo de orden, etc.) y del propio mensaje en sí. Después escribe este mensaje en su puerto de salida, que está conectado con el módulo *interpreteWifi*.

El hilo de recepción es el encargado también de la comprobación de que la conexión sigue establecida. En caso de no ser así, vuelve a establecer la conexión, mostrando el aviso.

5.4.Submódulo enviaDatos

La funcionalidad de este módulo es la de escoger que datos han de enviarse a la consola remota. Actualmente manda la posición y la orientación del robot para su monitoreo.

Este módulo lee de su puerto de entrada, que está conectado al módulo Robot_Posicion, una estructura *PosicionExtType.* De esta estructura extrae la posición y la orientación y los escribe con formato *MensajeType* en su puerto de salida, que está conectado con *servidorWifi.* La posición y la orientación son enviadas en mensajes independientes con cabeceras *posicion* y *orientacion* respectivamente. Esto hemos elegido hacerlo así por claridad, pero se podía haber enviado todo en un mismo mensaje. Habría que cambiar entonces la forma de interpretar los mensajes en la consola remota.

5.5.Submódulo interpreteWifi

Este módulo es el encargado de interpretar las órdenes que llegan al robot por parte de la consola remota. Su puerto de entrada está conectado con *servidorWifi* y su puerto de salida está conectado con *Cerebro*.

El módulo se encarga de leer de su puerto de entrada un *MensajeType.* Procesa este mensaje y dependiendo del tipo de cabecera realiza la acción correspondiente para tal mensaje. Esta acción puede ser establecer la posición, o la orientación, o decirle al robot si se encuentra en modo guía o en modo usuario, y en ese caso, qué dirección debe tomar. La orden es escrita en el puerto de salida, que está conectado con *cerebro*. Las ordenes se guardan temporalmente en una cola, para que *cerebro* tenga tiempo suficiente a leerlas y no perdamos ninguna orden, ya que esto podría resultar un fallo crítico.

5.6.Arquitectura y funcionamiento del sistema global

El sistema sigue las interfaces de comunicación con el pipeline, de modo que se crean tanto los sockets como las estructuras necesarias para guardar los mensajes al iniciar, a la vez que, según se haya instanciado a través de la configuración del XML que define el proyecto, abre la comunicación con las bibliotecas necesarias, en función de los argumentos de dicho XML.

Una vez creadas las estructuras, el módulo *enviaDatos* se encarga de recoger la posición y la orientación del robot. Escribe estos datos en su puerto de salida, que está conectado al de entrada de *servidorWifi*. El hilo de *servidorWifi* que se encarga de enviar los datos recoge estos del puerdo de entrada y los manda mediante la conexión de red. El hilo receptor, recibe las órdenes de la consola remota y las escribe en forma de mensaje en el puerto de salida de *servidorWifi*, que está conectado al puerto de entrada de *interpreteWifi*. El módulo *interpreteWifi* lee los mensajes de su puerto de entrada y los interpreta, mandando las órdenes al puerto correspondiente de *cerebro*.

En caso de que se hubiera perdido la conexión, esto sería detectado por *servidorWifi* y el socket se pondría de nuevo a la escucha hasta que la

consola remota se volviera a conectar, cosa que hace automáticamente al perder la conexión por motivos no deseados. La función de terminar de todos los módulos cierra los sockets y libera los recursos.

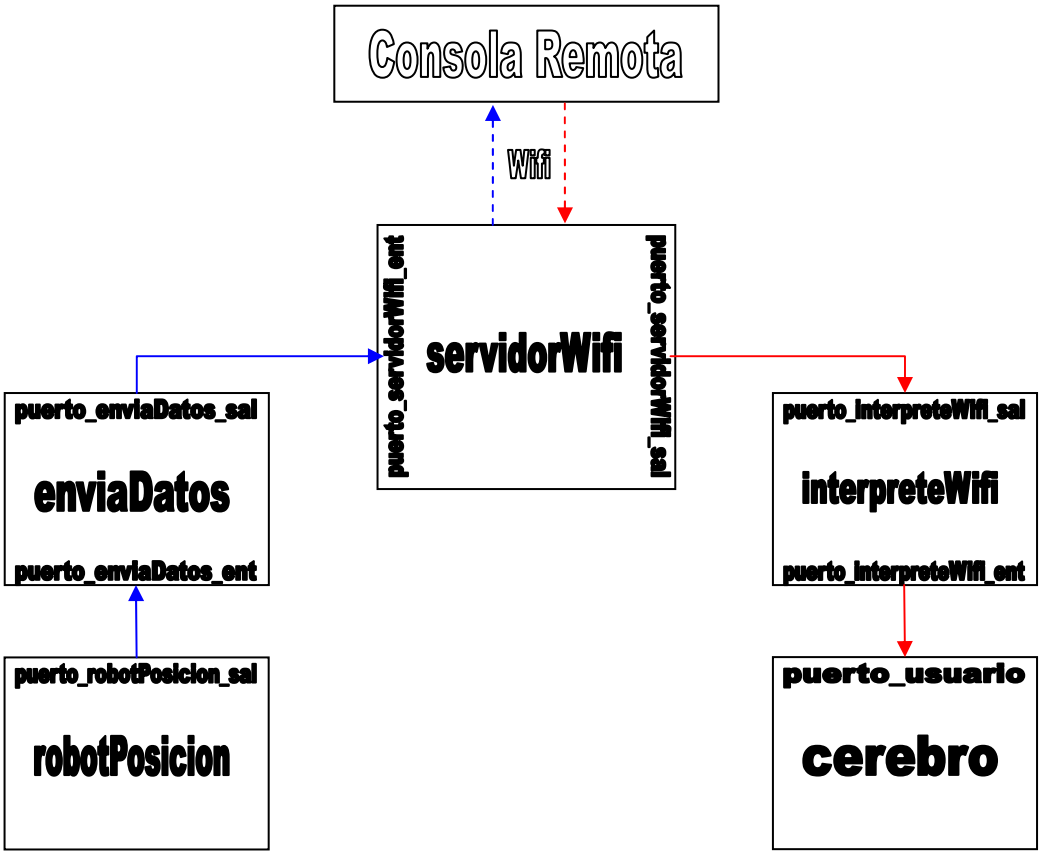


Fig 33. Esquema de la conexión de los módulos del sistema de comunicación

5.7.Bibliotecas

Hemos utilizado la biblioteca C++ Socket Class for Windows (*http://www.adp-gmbh.ch/win/misc/sockets.html*, (ref[7]) encontrada en internet. La elección de esta biblioteca ha sido por su sencillez y por la disposición del código fuente. Gracias a esto, hemos podido realizar modificaciones en la biblioteca original para conseguir funcionalidades necesarias para nuestro proyecto.

6. Librería Cargador de Xml

6.1.Introducción y descripción

Se trata de una librería estática que se encarga de transformar los ficheros xml de configuración de la aplicación en estructuras de datos manejables a nivel de programación.

Es utilizado por el modulo de sonido para obtener información del archivo xml de configuración de sonidos (ruta de los distintos archivos de sonido, identificador del sonido, duración). También lo utiliza el modulo lector de mapa, que necesita abstraer información del fichero xml que representa el mapa de entorno por el cual se mueve el robot (obstáculos, objetivos, paredes...).

6.2.Bibliotecas

Para la lectura de los distintos ficheros xml de configuración esta clase hace uso de un parser que suministra la biblioteca libxml++ incluida dentro de las librerías del paquete GTK.

7. Aplicación de edición de mapas de entorno

7.1.Introducción

El software que hemos desarrollado para el control del robot utiliza mapas del entorno por el que se moverá el robot, definidos como archivos xml. Este mapa es utilizado por el planificador para obtener la ruta libre de obstáculos que debe seguir el robot en cada momento.

Puesto que el museo de informática puede sufrir modificaciones importantes sobre la colocación de las distintas vitrinas en las que se exponen las distintas piezas, hemos desarrollado una pequeña aplicación realizada en Java y que sirve para editar gráficamente el mapa de entorno del museo, o cualquier otro escenario por el que se pueda mover el robot.

7.2.Funcionamiento

La aplicación está compuesta por un área de dibujo y un menú superior con las distintas opciones. El editor transforma los cambios realizados en el mapa gráfico en cambios correspondientes en el archivo xml asociado.

El usuario tiene las siguientes opciones:

7.2.1. Crear Xml

Mediante está opción se crea la estructura general del documento xml del mapa, sin tener aún ningún obstáculo ni objetivo definido. El usuario introduce los atributos del mapa con los cuales se configura el fichero xml asociado (ruta del fichero a crear, ruta de la dtd que define la estructura del xml, anchura y altura del mapa, escala de pixeles a cm reales,…) Ver Fig 34.

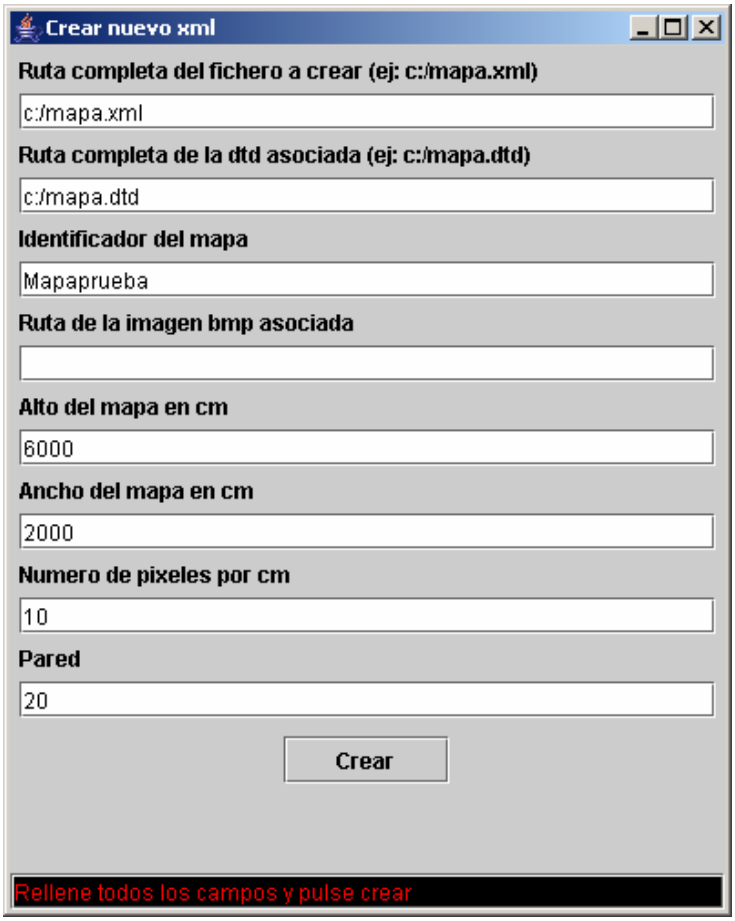


Fig 34. Ventana de creación de XML

7.2.2. Cargar Xml

Sirve para cargar un fichero xml descriptor de mapa ya existente, la aplicación muestra en el área de dibujo la representación gráfica del fichero. El usuario puede entonces modificar el archivo cargado.

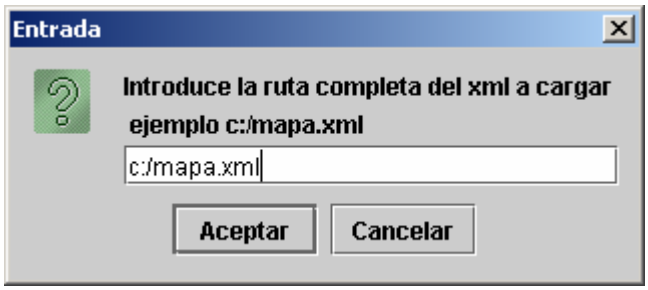


Fig 35. Ventana de cargar XML en el editor de XML

7.2.3. Modificar atributos del mapa

Mediante esta opción el usuario puede cambiar los atributos que definen el mapa y que se introdujeron al crear el fichero xml inicialmente.

7.2.4. Insertar obstáculo

Sirve para introducir un nuevo obstáculo en el mapa, definiendo sus vértices bien mediante clicks sobre el área de dibujo o introduciéndolos manualmente de forma numérica.

7.2.5. Insertar objetivos

Sirve para introducir un nuevo objetivo en el mapa, definiendo sus vértices bien mediante clicks sobre el área de dibujo o introduciéndolos manualmente de forma numérica.

7.2.6. Borrar

Seleccionando esta opción podemos borrar cualquier obstáculo u objetivo del mapa, simplemente haciendo click en su interior.

7.2.7. Guardar cambios

Se guardan en el fichero xml todos los cambios del mapa gráfico (área de dibujo) que se han realizado.

7.3.Ejemplos de mapas generados

Mostramos a continuación un ejemplo de mapa creado gráficamente y el fichero xml equivalente que crea el editor. Los objetos en rojo simbolizan obstáculos y los objetos en azul señalan objetivos en la misión del robot.

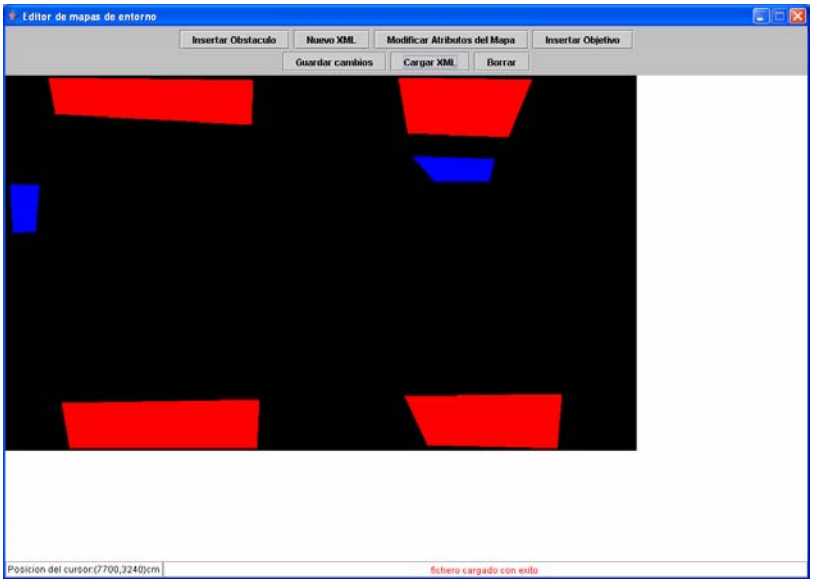


Fig 36. Mapa creado gráficamente por el usuario

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapa SYSTEM "mapa.dtd">
<mapa id="mapaEjemplo" ruta="" ancho="6000" alto="5000" pxcm="10" pared="20">
 <objetivos>
  <objetivo id="Vitrina1">
    <punto x="5150" y="1070"/>
    <punto x="6200" y="1110"/>
    <punto x="6140" y="1420"/>
    <punto x="5440" y="1410"/>
  </objetivo>
  <objetivo id="puerta">
    <punto x="50" y="1440"/>
    <punto x="430" y="1460"/>
    <punto x="380" y="2080"/>
    <punto x="90" y="2100"/>
  </objetivo>
 </objetivos>

<obstaculos>
  <obstaculo>
   <punto x="540" y="20"/>
   <punto x="3140" y="60"/>
   <punto x="3120" y="660"/>
   <punto x="630" y="510"/>
  </obstaculo><obstaculo>
   <punto x="4980" y="20"/>
   <punto x="6680" y="50"/>
   <punto x="6380" y="820"/>
   <punto x="5100" y="770"/>
 </obstaculo>
<obstaculo>
   <punto x="710" y="4370"/>
   <punto x="3220" y="4320"/>
   <punto x="3190" y="4980"/>
   <punto x="810" y="4970"/>
 </obstaculo>
<obstaculo>
   <punto x="5060" y="4280"/>
   <punto x="7060" y="4250"/>
   <punto x="7000" y="4980"/>
   <punto x="5350" y="4930"/>
 </obstaculo>
 </obstaculos>
 </mapa>
```

Fig 37. Fichero xml equivalente creado por el editor

7.4.Bibliotecas

Para la creación y modificación de archivos xml hemos utilizado la librería "Java API for XML Processing (JAXP)" (para más información consultar ref[8]).

Para la realización de la interfaz gráfica hemos usado la librería awt y swing de java.

8. Módulo de consola gráfica

8.1.Introducción

Se trata de un módulo compuesto por una ventana que permite interactuar a un usuario con el robot, para asignarle tareas de guía o para controlarlo manualmente mediante el teclado. Está pensado para configurar inicialmente el robot antes de que empiece a realizar labores de guía en el museo, indicándole manualmente en qué posición y orientación se encuentra en un principio. Este módulo se comunica con el módulo cerebro, indicándole las acciones que solicita mediante la consola el usuario que inicializa el robot y los diferentes parámetros de configuración que introduce.

8.2.Detalles del módulo

Entrada: Este módulo no recibe entradas del pipeline de la aplicación, las recibe directamente de un usuario mediante la interfaz gráfica que implementa .Estas entradas son comandos que definen el tipo de misión del robot (guía, control manual, modo presentación) y parámetros de inicialización (posición, orientación del robot, dirección a seguir en control manual...).

Salida: Escribe en su puerto de salida los distintos comandos filtrados que introduce el usuario en la consola y también los distintos parámetros de configuración establecidos para el robot.

Descripción: Módulo para la comunicación usuario-robot, podríamos decir que hace la función de "oído" del robot.

8.3.Bibliotecas

Para la creación de la interfaz gráfica de comunicación con el usuario utilizamos

la librería gráfica que ofrece GTK

8.4.Aspecto de la consola

Mostramos a continuación el aspecto que presenta la consola gráfica de comunicación con el robot.

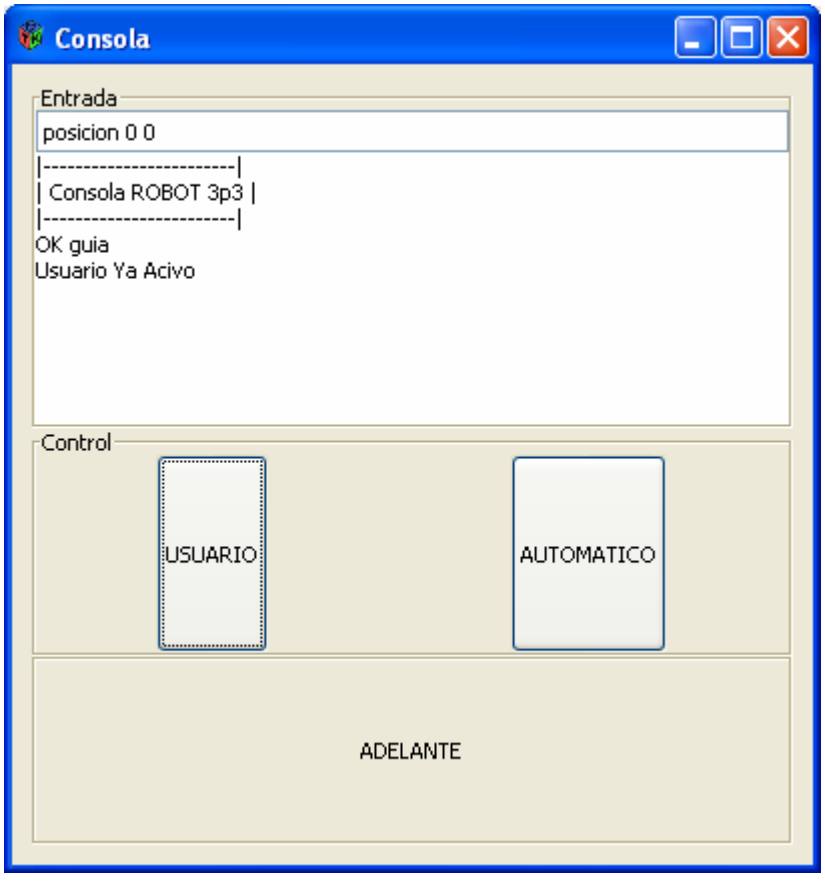


Fig 38. Consola gráfica

Capítulo 6

El sistema en funcionamiento

1. Introducción

En este capítulo se muestra el aspecto real que tiene el robot que hemos desarrollado. Asimismo mostramos de una forma visual un ejemplo del sistema funcionando, mediante fotos del robot ejecutando una misión de guía y las imágenes correspondientes de la simulación del robot con el mapa de entorno definido.

2. Aspecto físico y características de nuestro robot

Se trata de un robot circular con un radio de 19 cm y una altura incluyendo las ruedas de 48 cm. Utiliza motores con reductora 1:50 y encoders acoplados al eje de 500 pulsos. Dispone de sensores de infrarrojos y de ultrasonidos.As La alimentación se realiza mediante baterías de niquel-cadmio.

En las siguientes imágenes mostramos el aspecto físico de nuestra plataforma robótica.

Fig 39. Vista lateral-trasera del robot

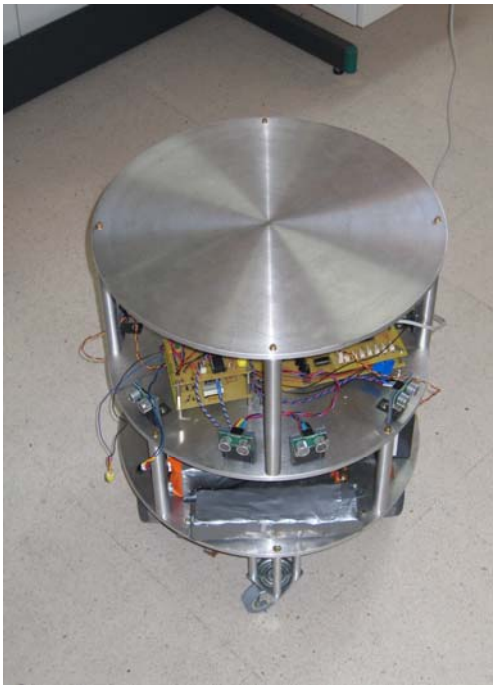


Fig 40. Vista superior-trasera del robot

3. Ejemplo de ejecución del sistema

3.1. Explicación del mapa de simulación

Para comprender mejor la simulación pasamos a explicar la ventana de simulación.

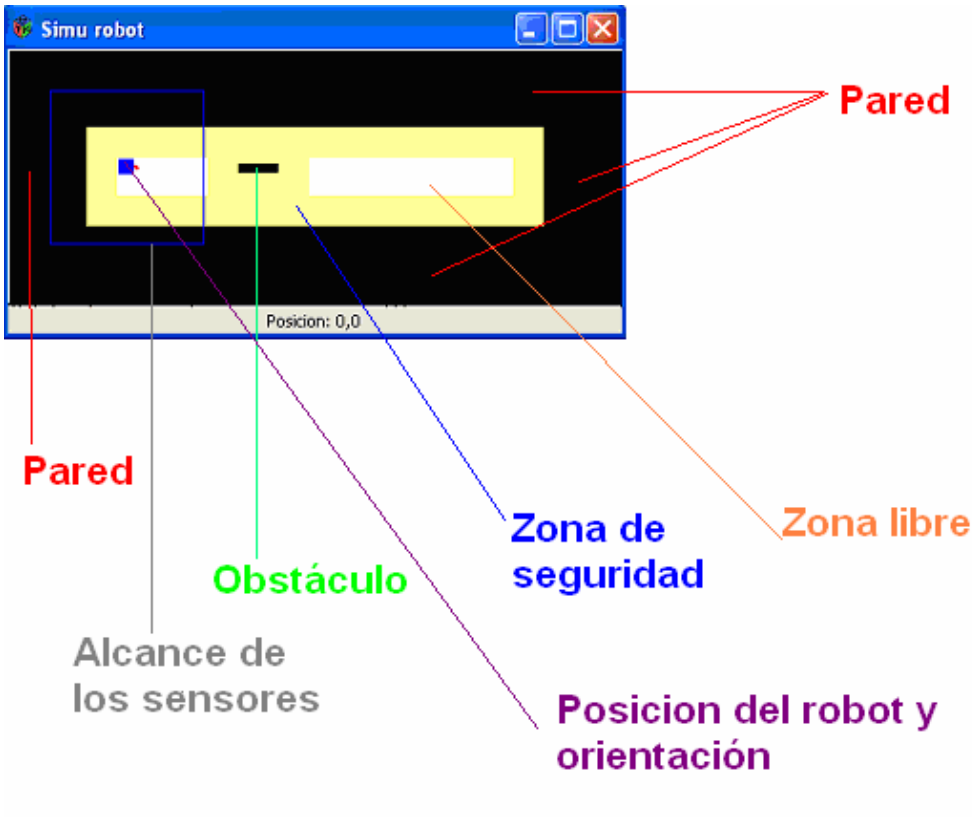


Fig 41. Explicación ventana de simulación

El cuadrado azul relleno simboliza la posición actual del robot, el punto rojo que sobresale de él señala la orientación que tiene el robot en ese momento. El cuadrado más grande azul y no relleno simboliza el radio de alcance de los sensores del robot. Los espacios en negro señalan paredes u obstáculos, los espacios blancos son espacios libres de obstáculos. Por simplicidad en la comprensión, en el ejemplo que

mostramos más adelante se usa este mapa, con un solo obstáculo. Las zonas en amarillo señalan el engrosado de los obstáculos y de las paredes, que pueden solaparse. Aunque en un principio parezca que el robot está atrapado en el rectángulo blanco de la izquierda, el sistema ante esta situación reduce la distancia de seguridad (siempre hasta un valor límite) para encontrar camino como se verá en el ejemplo de ejecución siguiente.

3.2.Desarrollo del ejemplo

A continuación mostramos imágenes de una ejecución de una misión de guía por parte del robot.

Lo primero que hacemos es indicar mediante la consola remota al robot de la posición inicial en la que se encuentra.

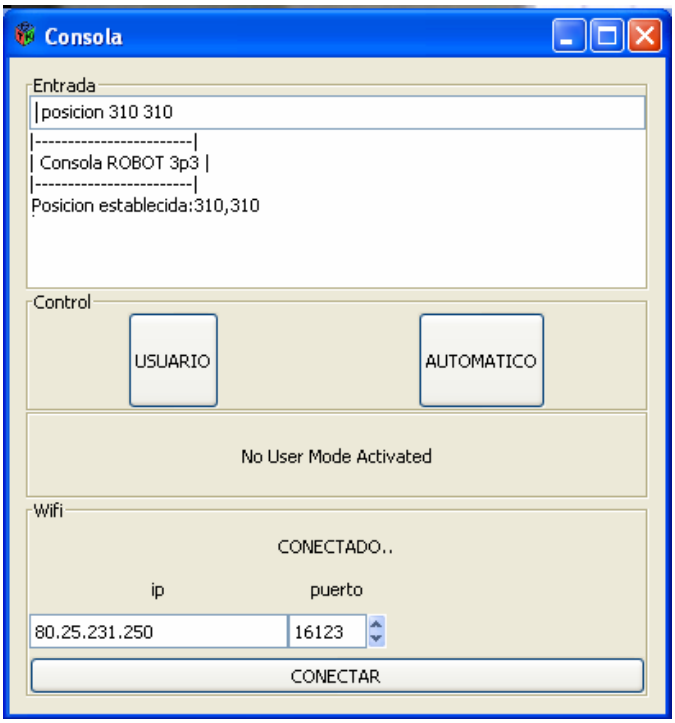


Fig 42. Establecimiento de la posición inicial en la consola remota

Seguidamente indicamos al robot que comience la guía mediante el comando "guía".

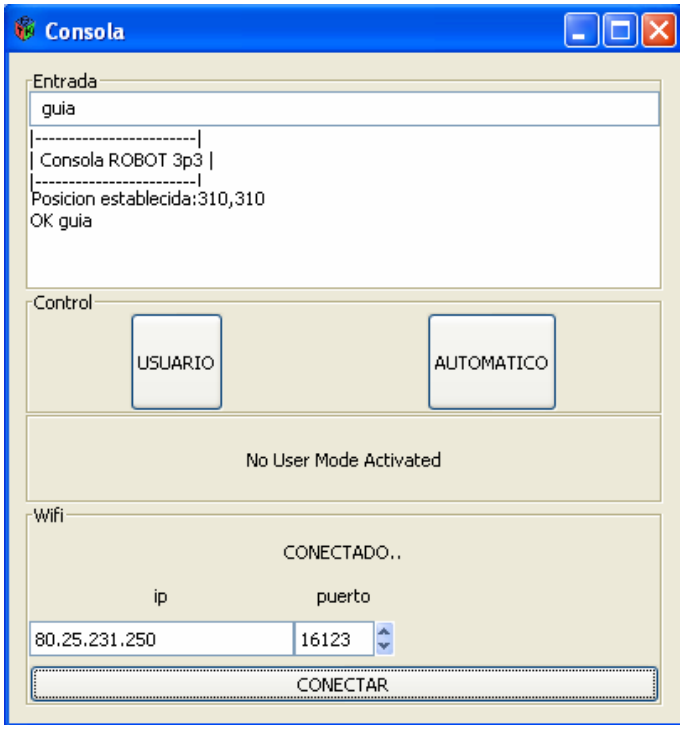


Fig 43.Llamada a la función de guía en la consola remota

A continuación mostramos la secuencia de navegación para esta guía.

Robot físico.

Situación inicial del robot real. Fig 44.

Fig 44. Secuencia navegación robot 1

Simulación robot.

Situación inicial de simulación. Fig 45.

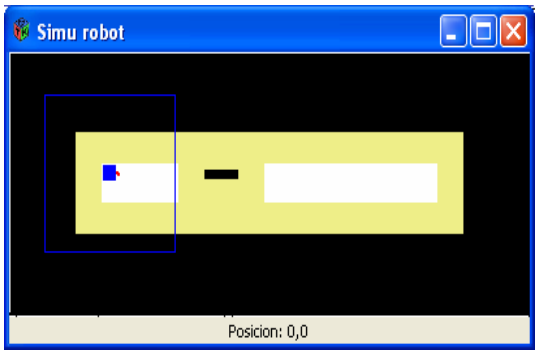


Fig 45. Secuencia simulación navegación 1

El robot llega a su primer objetivo, se para y reproduce el sonido asignado para ese objetivo. Fig 46.

Fig 46. Secuencia navegación robot 2

Situación en la simulación, robot en objetivo. Fig 47.

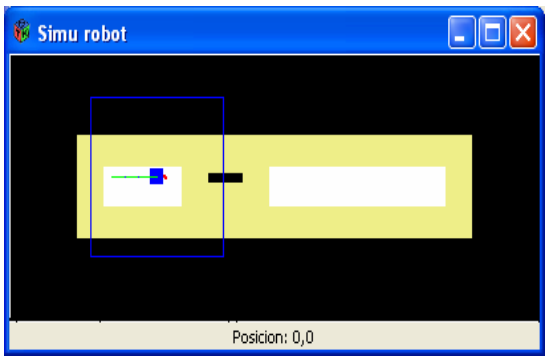


Fig 47. Secuencia simulación navegación 2

Debido a que la distancia de seguridad es muy grande el robot no podría seguir avanzando, sin embargo el sistema disminuye la distancia de seguridad (que tiene un límite mínimo) para que el robot encuentre camino y realiza una nueva replanificación. Fig 48.

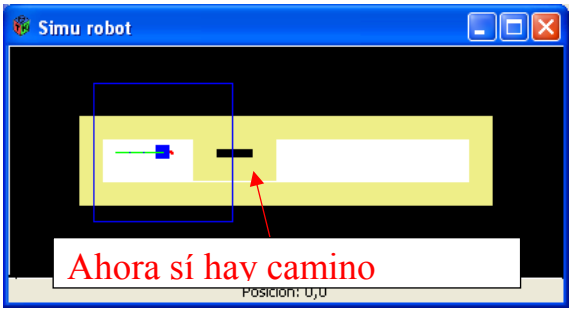


Fig 48. Disminución de distancia de seguridad y replanificación

El robot se desvía para esquivar el obstáculo. Fig 49.

Situación en la simulación, robot desviándose. Fig 50.

Fig 49. Secuencia navegación robot 3

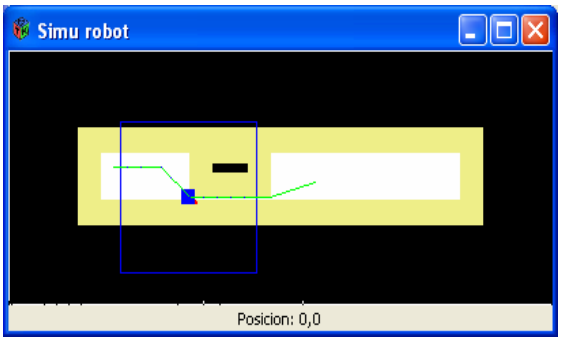


Fig 50. Secuencia simulación navegación 3

El robot se dirige hacia el siguiente objetivo. Fig 51. y Fig 52.

Fig 51. Secuencia navegación robot 4

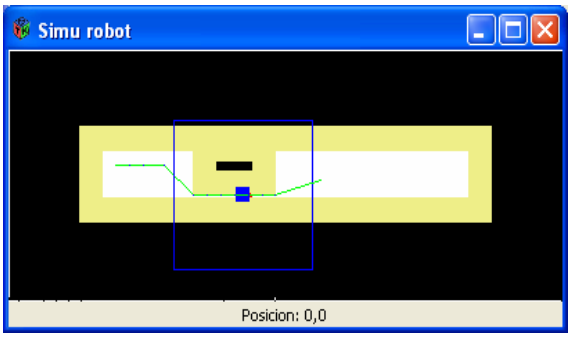


Fig 52. Secuencia simulación navegación 4

El robot llega a su segundo objetivo, se para y reproduce el sonido asignado para ese objetivo. Fig 53.

Situación en la simulación, robot en objetivo 2. Fig 54.

Fig 53. Secuencia navegación robot 5

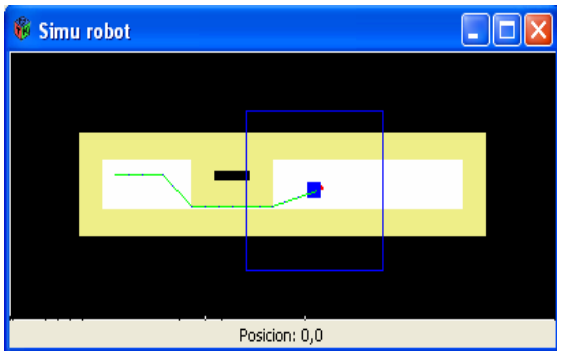


Fig 54. Secuencia simulación navegación 5

El robot llega al objetivo final donde se para. Fig 55. y Fig 56.

Fig 55. Secuencia navegación robot 6

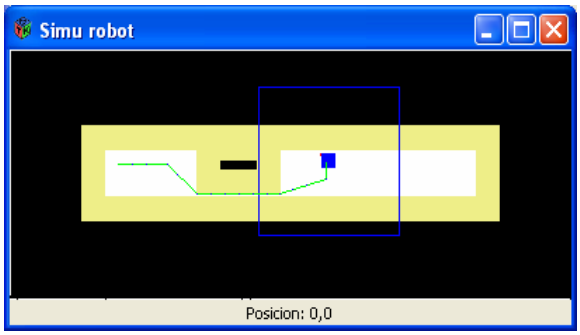


Fig 56. Secuencia simulación navegación 6

Aquí finaliza el ejemplo de ejecución del sistema, si se desean más ejemplos y pruebas remítase al cd de documentación que se suministra con esta memoria. Ahí podrá encontrar diversos videos de guías y fotos de nuestro robot.

4. Conclusiones

Desde que comenzamos y hasta la finalización completa del proyecto, hemos pasado por varias etapas. En una primera fase, afrontamos la aplicación como algo ligeramente borroso, no demasiado definido. Poco a poco fue tomando forma, proceso en el cual fuimos haciéndonos conscientes de algunos puntos clave en el desarrollo de un proyecto como este. Los más importantes que podemos esquematizar como conclusiones, son los que detallamos a continuación.

El proceso de desarrollo no ha de ser tomado como un bloque global que atacar directamente

Esto sólo lleva a un código confuso y muy acoplado, que da muy poca oportunidad a la expansión o a la reutilización del mismo, incluso dentro del proyecto. Hemos tenido la suerte de hacernos cargo de esta circunstancia en una fase poco avanzada del proyecto, e invertir un esfuerzo extra en organizar las cosas, dividir el trabajo a conciencia, con sentido, y asignando a cada cual las partes en las que más puede desarrollar sus habilidades. Asimismo, todo el tiempo invertido en realizar una buena arquitectura de trabajo, nos ha dado alas para ampliar la aplicación en la medida en que las necesidades del proceso fueron ampliándose.

El cumplimiento de los requisitos del proyecto ha de ser el objetivo primario

Con esto queremos señalar que, a pesar de que todo el proyecto se daba, como idea y como realidad, de una manera muy importante a la experimentación y a las pruebas, nuestra principal prioridad era cumplir con los requisitos primeros que estaban impuestos como fin del proceso de desarrollo.

En un proyecto a medio plazo como éste, ha sido muy importante la imposición de hitos sobre el cumplimiento de las distintos aspectos de la especificación. Una planificación adecuada es primordial para una finalización exitosa del proyecto.

La robótica móvil autónoma es un campo en el que queda mucho por hacer

Si bien es verdad que no hemos llegado ni mucho menos a lo más profundo, innovador e interesante de este campo, sí es cierto que podemos decir, tras haber estado casi un año experimentando y trabajando, que la robótica móvil autónoma tiene mucho por delante aún. A nuestro nivel nos hemos dado cuenta, por ejemplo, de lo muy sensible que es el calibrado de los parámetros reales de configuración del robot (distancia de seguridad, escalas de representación del entorno, dificultad para la localización espacial del robot sin utilizar balizas). Del funcionamiento teórico y simulado del sistema de control del robot a alto nivel al funcionamiento real en la plataforma física del robot hay un trabajo de sintonización e integración muy importante. La robótica móvil autónoma, según estos ejemplos y otros muchos que hemos encontrado es, a nuestro parecer, una ciencia que aún tiene que desarrollarse mucho y que estamos seguros dará resultados muy interesantes en un futuro próximo.

5. Agradecimientos

Gracias a Carlos León, compañero y alumno de la Facultad de Informática por proporcionarnos la arquitectura de tuberías con la que empezamos a trabajar y también por todas las buenas ideas que nos ha prestado a lo largo del proyecto.

Gracias también a todos los trabajadores del Laboratorio de Arquitectura de Computadores y Automática por prestarnos sus instalaciones y por las molestias que seguro les hemos causado.

Por último gracias a nuestros directores de proyecto profesor Dr. D. José Antonio López Orozco y profesor Dr. D. José Manuel Mendías Cuadros por haber tenido paciencia con nosotros.

Apéndice A

Métodos clásicos de planificación de trayectorias

1. Métodos clásicos de planificación

Todos ellos se fundamentan en una primera fase de construcción de algún tipo de

grafo sobre el espacio libre, según la información poseída del entorno, para posteriormente emplear un algoritmo de búsqueda en grafos que encuentra el camino óptimo según cierta función de coste.

1.1. Planificación basada en grafos de visibilidad

Los grafos de visibilidad (ref [11]) proporcionan un enfoque geométrico útil para resolver el problema de la planificación. Supone un entorno bidimensional en el cual los obstáculos están modelados mediante polígonos. Para la generación del grafo este método introduce el concepto de *visibilidad*, según el cual define dos puntos del entorno como *visibles* si y solo si se pueden unir mediante un segmento rectilíneo que no intersecte ningún obstáculo (si dicho segmento resulta tangencial a algún obstáculo se consideran los puntos afectados como visibles). En otras palabras, el segmento definido debe yacer en el espacio libre del entorno C_l.

Así, si se considera como nodos del grafo de visibilidad la posición inicial, la final y todos los vértices de los obstáculos del entorno, el grafo resulta de la unión mediante arcos de todos aquellos nodos que sean visibles.

En Fig 57 se muestra el grafo de visibilidad construido merced a los obstáculos poligonales existentes en el entorno y las configuraciones inicial q_a y final q_f.

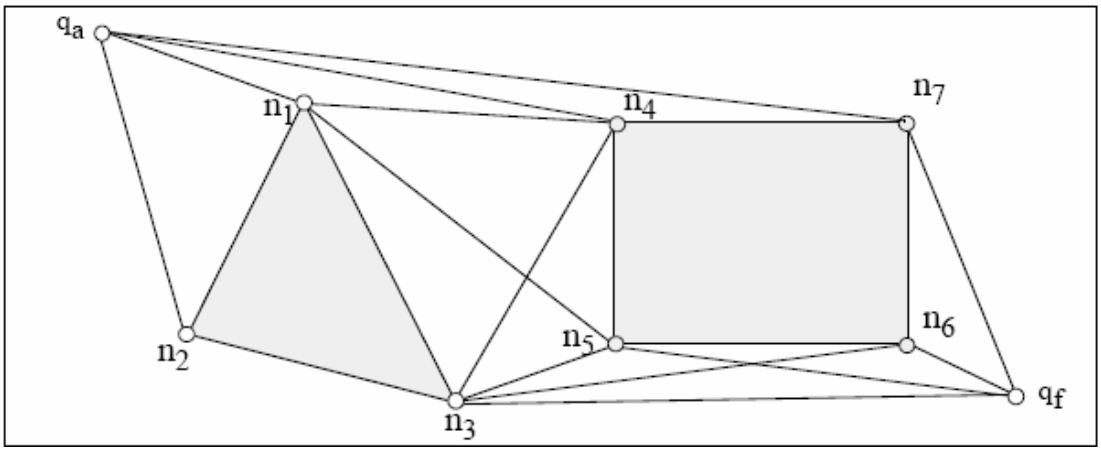


Fig 57. Grafo de visibilidad en un entorno de dos obstáculos

En el grafo mostrado (Fig 57.), se puede observar cómo sólo están unidos los nodos directamente visibles, de tal forma que el conjunto de arcos estará formado por las aristas de los obstáculos, más el resto de líneas que relacionan los vértices de los diferentes polígonos.

Posteriormente, y como explicamos en el módulo búsqueda de está sección, mediante un algoritmo de búsqueda en grafos se elige la ruta que una la configuración inicial con la final minimizando la distancia entre ambas. La ruta que cumple el objetivo de la navegación queda definida como una sucesión de segmentos que siguen los requisitos especificados.

1.2.Planificación basada en diagramas de Voronoi

Al contrario que los métodos basados en grafos de visibilidad, la planificación basada en diagramas de Voronoi sitúa la ruta lo más alejada posible de los obstáculos.

Con ello elimina el problema presentado por los grafos de visibilidad de construir rutas semilibres de obstáculos. Los diagramas de Voronoi se definen como una proyección del espacio libre del entorno en una red de curvas unidimensionales yacientes en dicho espacio libre.

Formalmente se definen como una retracción (ref [12]) con preservación de la continuidad. Si el conjunto C_l define las posiciones libres de obstáculos de un entorno, la función retracción *RT* construye un subconjunto C_v continuo de C_l.

De esta forma, se dice que existe un camino desde una configuración inicial q_a hasta otra final q_f, supuestas ambas libres de obstáculos, si y solo si existe una curva continua desde $RT(q_a)$ hasta $RT(q_f)$.

La definición de la función retracción *RT* implica la construcción del diagrama de Voronoi. La idea fundamental es ampliar al máximo la distancia entre el camino del robot y los obstáculos. Por ello, el diagrama de Voronoi resulta el lugar geométrico de las configuraciones que se encuentran a igual distancia de los dos obstáculos más próximos del entorno. El diagrama estará formado por dos tipos de segmentos: rectilíneos y parabólicos. La elección de la modalidad de segmento corresponde con la clase de elementos de los obstáculos más cercanos que se encuentren enfrentados entre sí. De esta forma, el lugar geométrico de las configuraciones que se hallan a igual distancia de dos aristas de dos obstáculos diferentes es una línea recta, mientras que en el caso de tratarse de un vértice y una arista resulta una parábola.

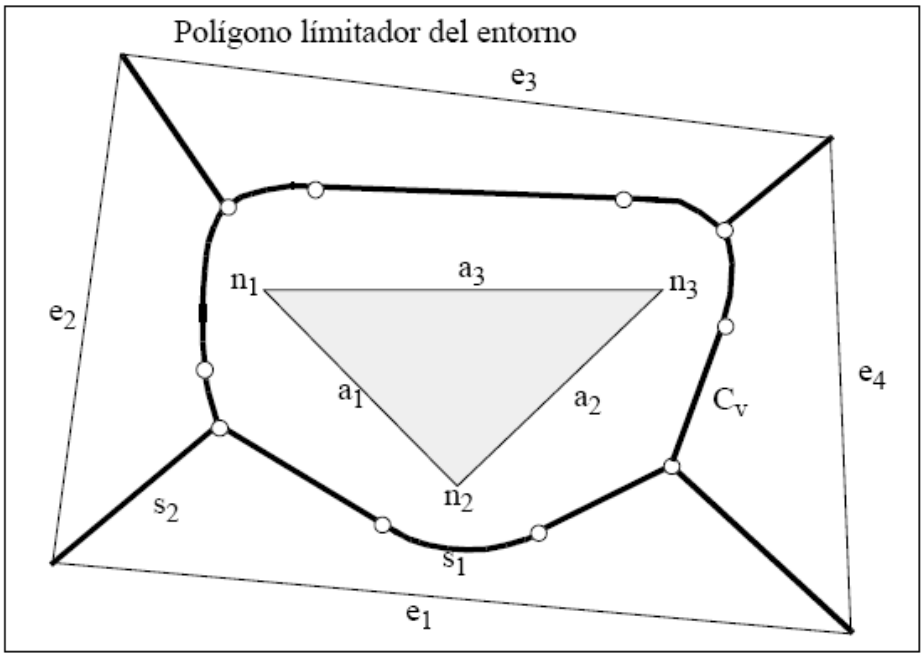


Fig 58. Retracción del espacio libre en un diagrama de Voronoi

En la Fig 58. se muestra un entorno delimitado por un polígono de aristas {e1,e2,e3,e4} y un obstáculo triangular de vértices {n1,n2,n3} y aristas {a1,a2,a3}. La retracción del espacio libre en una red continua de curvas es el diagrama de Voronoi Cv, representado mediante las líneas de trazo grueso. Los dos tipos de segmento utilizados en la construcción del

diagrama pueden distinguirse en la mencionada figura, así, el segmento s1 es el lugar geométrico de los puntos equidistantes entre la arista e1, y el vértice n2. Por otra parte, puede observarse como el segmento rectilíneo s2 cumple la misma condición pero con respecto a las aristas e1 y e2.

Dado una configuración *q* no perteneciente a C_v, existe un único punto *p* más cercano perteneciente a un vértice o arista de un obstáculo. La función *RT(q)* se define como el primer corte con C_v de la línea que une *p* con q (Fig 59.).

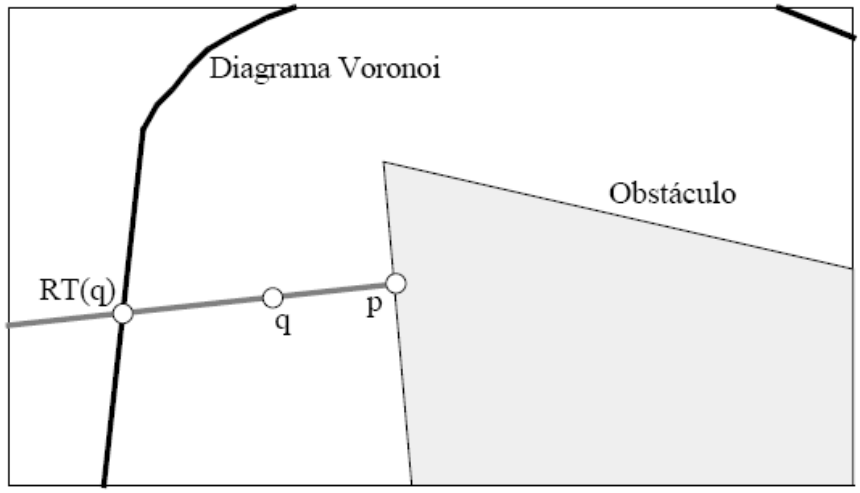


Fig 59. Imagen de una configuración q en el diagrama de Voronoi

El algoritmo de planificación, en esencia, consiste en encontrar la secuencia de segmentos *Si* del diagrama de Voronoi tal que conecten $RT(q_a)$ con $RT(q_f)$, Dicha secuencia conforma la ruta buscada. A continuación se describe el algoritmo:

1. Calcular el diagrama de Voronoi.
2. Calcular $RT(q_a)$ y $RT(q_f)$.
3. Encontrar la secuencia de segmentos $\{s_1,...,s_p\}$ tal que $RT(q_a)$ pertenece a s_1 y $RT(q_f)$ pertenece a s_p.
4. Si se encuentra dicha secuencia, devolver la ruta. Si no indicar condición de error.

Al igual que los grafos de visibilidad, este método también trabaja en entornos totalmente conocidos y con obstáculos modelados mediante polígonos.

1.3.Planificación basada en modelado del espacio libre

Se aplica a arquetipos de entornos con obstáculos poligonales, y la planificación en este caso se realiza mediante el modelado del espacio libre (ref [3]). Esta acción se lleva a cabo por los denominados cilindros rectilíneos generalizados (*CRG*). Al igual que los diagramas de Voronoi, con el uso de los *CRG* se pretende que el vehículo navegue lo más alejado de los obstáculos. De forma que la ruta que lleve al robot desde una configuración inicial hasta otra final estará compuesta por una serie de *CRG* interconectados, de tal modo que la configuración de partida se encuentre en el primer cilindro de la sucesión y la final en el último.

La construcción de un *CRG* se realiza a partir de las aristas de los distintos obstáculos que se encuentran en el entorno. Para que un par de aristas $_1a_i$ y $_2a_j$ pertenecientes a los obstáculos b_1 y b_2 respectivamente puedan formar un cilindro generalizado, deben cumplir las siguientes condiciones:

- La arista $_1a_i$ está contenida en una recta que divide al plano en dos regiones.
    - La arista $_2a_j$ debe yacer por completo en la región opuesta en la que se
    - encuentra situada b_1. Este criterio es simétrico.
- El producto escalar de los vectores normales con dirección hacia el exterior del obstáculo que contiene cada arista debe resultar negativo.

Si se cumplen estas condiciones significa que ambas aristas se encuentran enfrentadas, y por tanto se puede construir un CRG con ellas (Fig 60.).

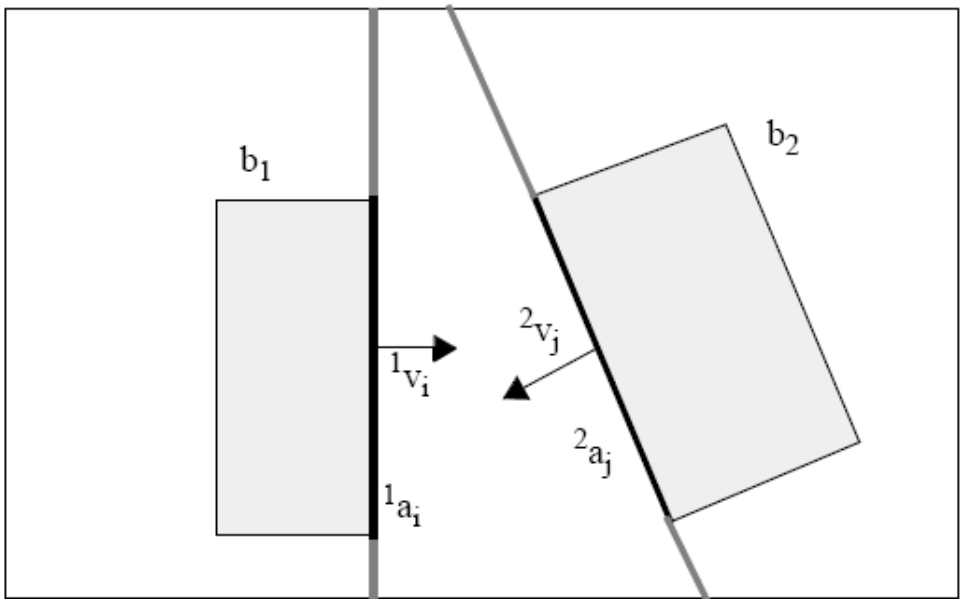


Fig 60. Condiciones que deben cumplir dos aristas para construir un CRG

Una vez detectadas dos aristas que pueden formar un CRG, el siguiente paso será construirlo. El proceso para alcanzar este cometido, se encuentra descrito en Fig 61.

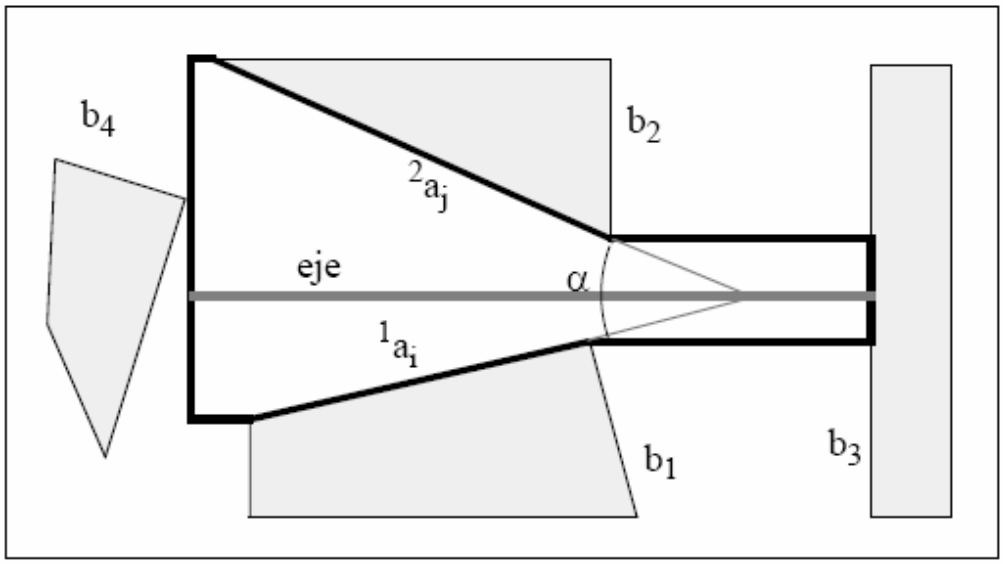


Fig 61. Construcción de un CRG

El primer paso es el cálculo del *eje* del *CRG*, el cual se define como la bisectriz del ángulo α formado por el corte de las rectas que contienen las aristas 1a_i y 2a_j que cumplen las condiciones expuestas anteriormente. Por ambos lados de dichas aristas se construyen segmentos rectilíneos paralelos al *eje*, con origen en los vértices de las aristas implicadas y con extremo señalado por la proyección del primer obstáculo que corta el *eje*.

Repitiendo este proceso, se construye una red *CRG* en el entorno del robot que modela el espacio libre del mismo. El robot navegará por el eje del cilindro, en el cual se encuentran anotadas para cada punto el rango de orientaciones admisibles. El paso de un *CRG* a otro se produce siempre y cuando sus ejes intersecten y la intersección del rango de orientaciones admisibles en el punto de corte de ambos *ejes* no sea nulo.

1.4.Planificación basada en la descomposición en celdas

Este tipo de métodos se fundamenta en una descomposición en celdas del espacio libre (ref [12]). Así, la búsqueda de una ruta desde una postura inicial q_a hasta otra final q_f, consiste en encontrar una sucesión de celdas que no presente discontinuidades, tal que la primera de ellas contenga a q_a y la última a q_f. Al contrario que los métodos expuestos a lo largo de este apartado, no encuentra una serie de segmentos que modele la ruta, sino una sucesión de celdas; por ello, se hace necesario un segundo paso de construcción de un *grafo de conectividad*, encargado de definir la ruta.

Para la planificación según el método de descomposición en celdas, se precisa la resolución de dos problemas: la descomposición del espacio libre en celdas y la construcción de un grafo de conectividad. El primero de ellos implica construir unas celdas con determinada forma geométrica tal que resulte fácil de calcular un camino entre dos configuraciones distintas pertenecientes a la celda, y la comprobación para averiguar si dos celdas son adyacentes debe disfrutar de la mayor simpleza posible. Aparte de estas características, la descomposición global del espacio libre implica que no deben existir solapamientos entre celdas y que la unión de todas ellas corresponde exactamente al espacio libre.

El grafo de conectividad es un grafo no dirigido, y su construcción está asociada a la descomposición en celdas efectuada en el paso anterior, del tal forma, que los nodos van a ser cada una de las celdas, existiendo un arco entre dos celdas si y solo si son adyacentes.

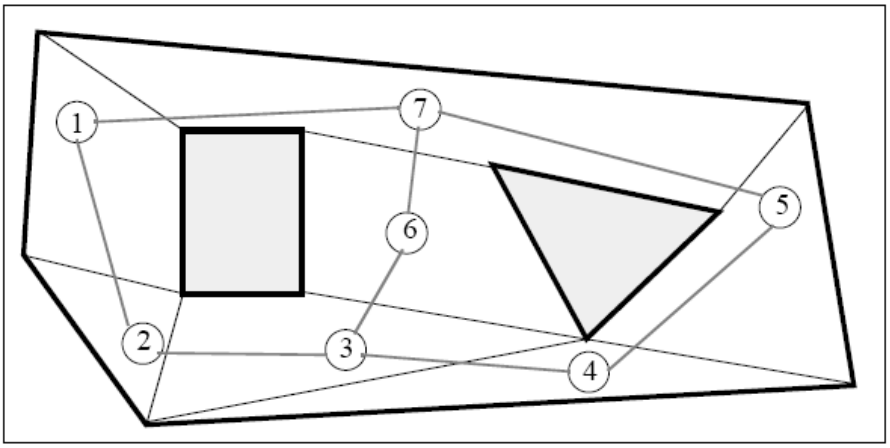


Fig 62. Descomposición en celdas y grafo de conectividad

Los distintos métodos basados en este principio, se distinguen por la forma en la cual realizan la descomposición en celdas y como se construye el grafo de conectividad. El método más sencillo de descomposición del espacio libre del entorno en celdas resulta el denominado descomposición trapezoidal. Este método se basa en la construcción de segmentos rectilíneos paralelos al eje Y del sistema global F_g a partir de los vértices de cada uno de los elementos del entorno. El final del segmento queda delimitado por el primer corte de la línea con un elemento del entorno. Esta descomposición es la mostrada en Fig 63.

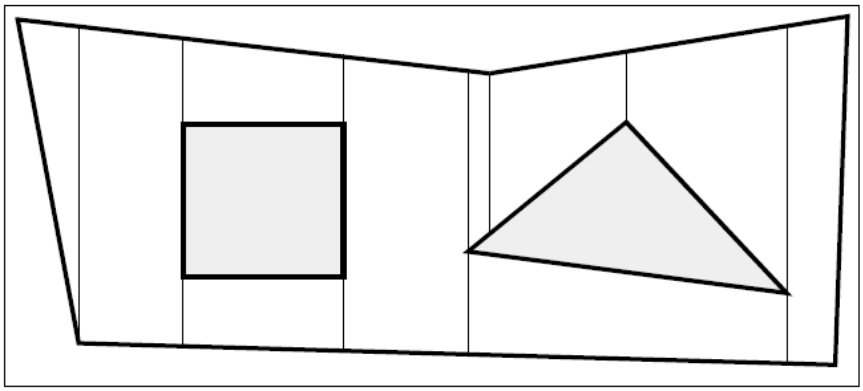


Fig 63. Descomposición trapezoidal del espacio libre

El grafo de conectividad se construye por medio de la unión de los puntos medios de los segmentos verticales definidos (Fig 64.).

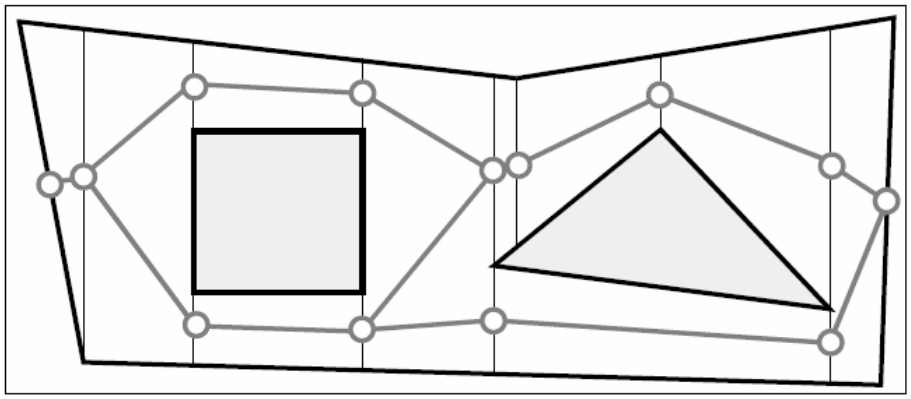


Fig 64. Grafo de conectividad de una descomposición trapezoidal

Este tipo de enfoque se presta a muchas variantes, por ejemplo la utilización de varios niveles de resolución para una búsqueda jerarquizada, o bien el uso de celdas en tres dimensiones para la planificación de caminos en espacios tridimensionales.

Apéndice B

Ejemplo de ficheros xml de configuración del sistema

Estas configuraciones se leen al ejecutar la aplicación y no necesita compilación. Por lo tanto es una de las herramientas más potentes para los ajustes y modificaciones.

A continuación se muestra un fichero de configuración utilizado por la aplicación.

Para la simulación del robot solo tiene entradas conectadas a los módulos del alto-nivel: Cerebro y planificador Local, y una salida: La posición.

```
<!-- Simulacion -->
<modulo nombre="simu_robot" ruta="simu_robot" estado="activo">
  <conexion origen="puerto_usuario" destino="puerto_directo"
cola="4">
cerebro
  </conexion>
  <conexion origen="puerto_movimiento"
destino="puerto_movimientos" cola="4">
planificador_local
  </conexion>
  <conexion origen="puerto_velocidad" destino="puerto_velocidad"
cola="4">
planificador_local
  </conexion>
  <salida puerto="puerto_posicion">2</salida>
</modulo>
```

Para la simulación del entorno podemos modificar el rango de los sensores, la probabilidad de que aparezcan más obstáculos y el tiempo de ciclo.

```
<modulo nombre="simu_entorno" ruta="simu_entorno" estado="activo">

  <argumento nombre="radio_sensores">200</argumento>
  <argumento nombre="probabilidad">0</argumento>
  <argumento nombre="tiempo">1000000</argumento>
  <conexion origen="puerto_posicion"
destino="puerto_robot_pos_sal" cola="4">
Robot_Posicion
  </conexion>
  <salida puerto="puerto_obstaculos_entorno_sal">4</salida>
</modulo>
```

En La ventana local podemos definir el tamaño del comando

```
<!-- VENTANA LOCAL -->
<modulo nombre="consolaVentana" ruta="consolaVentana"
estado="activo">
  <argumento nombre="tam_comando">20</argumento>
  <salida puerto="puerto_salida">2</salida>
</modulo>

<!-- NÚCLEO-->
```

El Mapa se define por los parámetros de seguridad engrosado y centímetros por celda para el enrejado.

Podemos decidir si se captan obstáculos o no y si funciona el servicio de peticiones

Las rutas del mápa y de las rutas también se pueden configurar.

```
<modulo nombre="lectormapa" ruta="lectormapa" estado="activo">
  <argumento nombre="engrosado">20</argumento>
  <argumento nombre="seguridad">60</argumento>
  <argumento nombre="celdacm">10</argumento>
  <argumento nombre="ruta_mapa_xml">xml/mapa.xml</argumento>
  <argumento nombre="ruta_guia_xml">xml/mapaGuia.xml</argumento>
  <argumento nombre="acepta_peticiones">1</argumento>
  <argumento nombre="acepta_obstaculos">1</argumento>
  <argumento nombre="factor_olvido">10</argumento>

  <conexion origen="puerto_mapa_actualizar_obstaculo"
destino="puerto_obstaculos_local_sal" cola="10">
            planificador_local
  </conexion>
  <conexion origen="puerto_mapa_peticion"
destino="puerto_peticiones_mapa" cola="2">
            cerebro
  </conexion>
  <conexion origen="puerto_mapa_peticion"
destino="puerto_peticiones_mapa" cola="2">
            planificador
  </conexion>
  <conexion origen="puerto_mapa_peticion"
destino="puerto_peticiones_mapa" cola="2">
            planificador_local
  </conexion>
  <salida puerto="puerto_mapa">2</salida>
      <salida puerto="puerto_mapa_objetivos">2</salida>
      <salida puerto="puerto_mapa_objetivos_obstaculo">2</salida>
      <salida puerto="puerto_mapa_obstaculos">2</salida>
      <salida puerto="puerto_mapa_zona">2</salida>
      <salida puerto="puerto_mapa_obs_debug">2</salida>

</modulo>
```

En el cerebro se identifica el tiempo de ciclo, el tiempo de ejecución de una acción antes de llegar al objetivo y el de espera después de llegar al objetivo. También la precisión para llegar al objetivo y el error permitido de estimación de llegada.

Están desconectados los puertos que usan la simulación y si estan conectados a bajo nivel: Por ejemplo la posición.

```
<modulo nombre="cerebro" ruta="cerebro" estado="activo">
  <argumento nombre="tiempo_accion_i">0</argumento>
  <argumento nombre="tiempo_accion_f">0</argumento>
  <argumento nombre="margen_objetivo">20</argumento>
  <argumento nombre="error_estimacion_tiempo">1</argumento>
  <argumento nombre="tiempo_ciclo">500000</argumento>

<!--<conexion origen="puerto_usuario"
destino="puerto_interprete_wifi_sal"
cola="2">interpreteWifi</conexion>-->
      <conexion origen="puerto_usuario" destino="puerto_salida"
cola="2">consolaVentana</conexion>
  <conexion origen="puerto_mapa" destino="puerto_mapa_objetivos"
cola="2">lectormapa</conexion>
<!--<conexion origen="puerto_posicion" destino="puerto_posicion"
cola="2">simu_robot</conexion>-->
      <conexion origen="puerto_posicion"
destino="puerto_robot_posicion_sal"
cola="10">Robot_Posicion</conexion>
      <conexion origen="puerto_velocidad"
destino="puerto_velocidad" cola="4">planificador_local</conexion>
      <conexion origen="puerto_accion"
destino="puerto_sonido_duracion" cola="2">salidaSonido</conexion>

      <salida puerto="puerto_trayectoria">2</salida>
            <salida puerto="puerto_timer">2</salida>
            <salida puerto="puerto_directo">4</salida>
            <salida puerto="puerto_texto">4</salida>
            <salida puerto="puerto_sonido">4</salida>
            <salida puerto="puerto_peticiones_mapa">2</salida>
            <salida puerto="puerto_posicion_salida">2</salida>
            <salida puerto="puerto_planificador_BN">10</salida>
            <salida puerto="puerto_velocidad_salida">4</salida>

</modulo>
```

El planificador Global es muy sencillo. Parámetros: número máximo de replanificaciones. Conexiones: las entradas están conectadas a mapa, cerebro (trayectorias), Planificador Local (trayectorias para replanificar)

```
<modulo nombre="planificador" ruta="planificador" estado="activo">
      <argumento nombre="max_replan">3</argumento>
          <conexion origen="puerto_mapa"
destino="puerto_mapa_obstaculos" cola="2">lectormapa</conexion>
        <conexion origen="puerto_entrada"
destino="puerto_trayectoria" cola="2">cerebro</conexion>
         <conexion origen="puerto_entrada" destino="puerto_replan"
cola="2">planificador_local</conexion>
        <salida puerto="puerto_texto">4</salida>
        <salida puerto="puerto_salida">4</salida>
</modulo>
```

El planificador Local es el más complejo y más ajustable. Parámetros: radio de los sensores, desviación máxima, número máximo de replanificaciones, la distancia de solapado. Dist_Punto: Es un parámetro muy importante. Define cuanta separación hay entre los puntos planificados que se mandan al robot. Podemos así cambiarlo dependiendo de nuestro entorno, y simplemente rearrancando el robot. Interrupciones: Vemos un ejemplo de uso de las interrupciones. El planificador local acepta interrupciones del cerebro, por ejemplo para pararlo todo cuando el usuario pide el control remoto.

```
    <modulo nombre="planificador_local" ruta="planificador_local"
estado="activo">
          <argumento nombre="debug">0</argumento>

      <argumento nombre="dist_punto">40</argumento>
      <argumento nombre="radio_sensores">200</argumento>
      <argumento nombre="radio_busqueda">300</argumento>
      <argumento nombre="diff_solapado">50</argumento>
      <argumento nombre="margen_solapado">10</argumento>
      <argumento nombre="margen_objetivo_x">20</argumento>
      <argumento nombre="margen_objetivo_y">20</argumento>
      <argumento nombre="distancia_obj_min">40</argumento>
      <argumento nombre="desvio_maximo">150</argumento>
      <argumento nombre="dist_seguridad">60</argumento>
      <argumento nombre="radio_robot">20</argumento>
      <argumento nombre="max_replan">3</argumento>
      <argumento nombre="tam_lista_obj_punto">20</argumento>

          <!-- Diferenciar entre Simulacuion y Robot Entorno -->
       <!--   <conexion origen="puerto_obstaculos_ent"
destino="puerto_obstaculos_sal"
cola="10">robot_entorno</conexion>-->
          <conexion origen="puerto_obstaculos_ent"
destino="puerto_obstaculos_entorno_sal"
cola="10">simu_entorno</conexion>
          <conexion origen="puerto_mapa"
destino="puerto_mapa_zona" cola="2">lectormapa</conexion>
<!--        <conexion origen="puerto_posicion"
destino="puerto_posicion" cola="10">simu_robot</conexion>-->
        <conexion origen="puerto_posicion"
destino="puerto_robot_posicion_sal"
cola="10">Robot_Posicion</conexion>
        <conexion origen="puerto_entrada" destino="puerto_salida"
cola="4">planificador</conexion>
        <salida puerto="puerto_replan">4</salida>
        <salida puerto="puerto_movimientos">20</salida>
        <salida puerto="puerto_velocidad">10</salida>
        <salida puerto="puerto_imagen">10</salida>
        <salida puerto="puerto_obstaculos_local_sal">10</salida>
        <salida puerto="puerto_peticiones_mapa">5</salida>
        <interrupcion>cerebro</interrupcion>
</modulo>
```

Entorno 2D, está conectadoa todas las salidas de los módulos sin afectar al funcionamiento si se desconectao no.

```
<modulo nombre="salidaImagen" ruta="salidaImagen" estado="activo">

      <argumento nombre="mostrar_todo">1</argumento>
      <argumento nombre="pixcm">10</argumento>
      <argumento nombre="radio_sensores">200</argumento>
      <argumento nombre="radio_robot">20</argumento>
      <argumento nombre="tiempo_refresco">500000</argumento>

<!--        <conexion origen="puerto_posicion"
destino="puerto_posicion" cola="4">simu_robot</conexion>-->
            <conexion origen="puerto_posicion"
destino="puerto_robot_posicion_sal"
cola="4">Robot_Posicion</conexion>
            <conexion origen="puerto_local"
destino="puerto_imagen" cola="4">planificador_local</conexion>
            <conexion origen="puerto_trayectoria"
destino="puerto_movimientos"
cola="4">planificador_local</conexion>
            <conexion origen="puerto_global"
destino="puerto_salida" cola="4">planificador</conexion>
            <!-- Diferenciar entre Simulacuion y Robot Entorno -->
    <conexion origen="puerto_obstaculos_ent"
destino="puerto_obstaculos_sal" cola="10">robot_entorno</conexion>
<!--        <conexion origen="puerto_obstaculos_ent"
destino="puerto_obstaculos_entorno_sal"
cola="4">simu_entorno</conexion>-->
      <!--  <conexion origen="puerto_obstaculos_ent"
destino="puerto_mapa_obs_debug" cola="4">lectormapa</conexion>
      -->
            <conexion origen="puerto_mapa"
destino="puerto_mapa_zona" cola="4">lectormapa</conexion>
    </modulo>
```

El sonido carga una lista con las rutas e identificadores de los sonidos

```
<modulo nombre="salidaSonido" ruta="salidaSonido" estado="activo">

  <argumento nombre="ruta_sonido_xml">xml/sonidos.xml</argumento>

      <conexion origen="puerto_sonido_salida"
destino="puerto_sonido" cola="4">
            cerebro
  </conexion>

  <salida puerto="puerto_sonido_duracion">10</salida>
</modulo>
```

Apéndice C

Índice de ilustraciones

Índice de ilustraciones

Bibliografía

[1] K.S. Fu, R.C. González, C. S. G. Lee "Robótica: control, detección, visión e inteligencia" McGraw-Hill

[2] Phillip John McKerrow "Introduction to Robotics" Addison-Wesley

[3] Brooks R. A. (1.983) "Solving the Find-Path Problem by Good Representation of Free Space". IEEE Transactions on Systems, Man and Cybernetics. Vol. 13, pp-190-197.

[4] Nelson, L y Cox, I. "Local Path Control for an Autonomous Vehicle".Springer-Verlag ,1.990.

[5] Russell, S. y Norvig, P. "Artificial Intelligence: A Modern Approach" Prentice Hall, 2004.

[6] Rich, E. y Knight, K. "Artificial Intelligence" McGraw-Hill, 1994.

[7] Quinn, B. y Shute, D. "Windows Sockets Network Programming" Addison-Wesley, 1995.

[8] Ahmed, K. Ancha, S. y otros. "Professional Java xml" Wrox, 2001.

[9] Levi P. "Principles of Planing and Control Concepts for Autonomous Mobile Robots". Proc. of 1987 IEEE International Conference on Robotics and Automation. pp 874-881. 1.987

[10] Lozano-Pérez T. "Spatial Planning: A Configuration Approach". IEEE Transactions on Computers. Vol. 32, pp 108-120. 1.983

[11] Nilsson N.J. " A Mobile Automaton: An Application of Artificial Intelligence Techiques". Proc. of the 1st. International Joint Conference on Artificial Intelligence. pp 509-520. 1.969

[12] Janich K. "Topology". Springer-Verlag, New-York. 1.984

[13] Thorphe C. "FIDO: Vision and Navigation for a Robot Rover". Ph. D. Thesis,Carnegie Mellon University. 1.984

[14] Borenstein J., Koren Y. (1.989) "Real-Time Avoidance for Fast Mobile Robots". IEEE Transactions on Systems, Man, and Cibernetics. Vol. 19, No 5, septiembre/octubre de 1.989. pp 1.179-1.234.

[15] Marsh V. "Un insecto robot logra camuflarse en una colonia de cucarachas". Diario de ciencia y tecnología La Flecha.

[16] Morales R. "Crean un robot que fabrica una casa en un día y sin intervención humana". Revista electrónica Tendencias Tecnológicas (www.tendencias21.net)

[17] Egger Mancilla J. "Desarrollo de interfaces sensibles a las emociones" Ph. D. Thesis, Universidad de Chile. 2002.

[18] http://www.gtkmm.org/docs/gtkmm-2.4/docs/