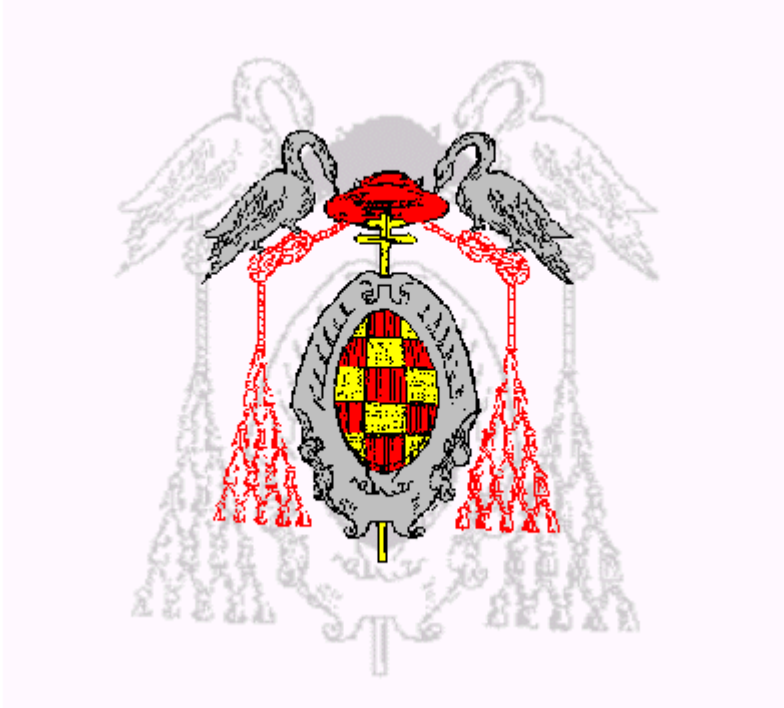
UNIVERSIDAD DE ALCALÁ

Escuela Politécnica

INGENIERÍA EN ELECTRÓNICA

Intensificación en Sistemas Electrónicos de Automatización y Control

Trabajo Fin de Carrera



NAVEGACIÓN AUTÓNOMA DE UNA SILLA DE RUEDAS EN INTERIORES PARCIALMENTE ESTRUCTURADOS.

Marta Marrón Romera.

Septiembre de 2000.

UNIVERSIDAD DE ALCALÁ

Escuela Politécnica

INGENIERÍA EN ELECTRÓNICA

Intensificación en Sistemas Electrónicos de Automatización y Control

Trabajo Fin de Carrera

NAVEGACIÓN AUTÓNOMA DE UNA SILLA DE RUEDAS EN INTERIORES PARCIALMENTE ESTRUCTURADOS.

Autora: Marta Marrón Romera.
Tutor: Juan Carlos García García.

TRIBUNAL:

Presidente: D. Felipe Espinosa Zapata.

Vocal 1º: D. Miguel Ángel Sotelo Vázquez.

Vocal 2º: D. Juan Carlos García García.

CALIFICACIÓN.....

FECHA.....

En lo personal.....

A Rubén, Nedi y Miguel Angel, simplemente por ser la constante que siempre está ahí y lo soporta todo, animando cuando es necesario y recriminando cuando es imprescindible (también lo he merecido).

A Julia y Ángela, por haber sufrido estoicamente el tener que ir a la pisci solas y ver marcada su niñez por una prima que siempre “está en la universidad”.

A Emilio, por ..... buff.... El ha sido la lanzadera a un nuevo mundo y a la vez la órbita en la que girar que tanto necesitaba.

A los pocos amigos de siempre que van quedando, por seguir intentando mantener la relación preguntando por el tfc, aunque la distancia vaya creciendo con las canas, bodas, casas y demás “asuntos insignificantes”. Gracias, Montse, Verónica, Marijose &, Josemi y, los chicos, que tienen sus momentos.

En lo profesional....

A Afredo, Michael y Elena que con sus trabajos y su continua preocupación han sido mis introductores en el mundo de la navegación, junto a mi tutor y mentor, Juan Carlos.

.... y a Emilio....

Y en definitiva.....

... los compañeros de curro siempre son parte importante de tu vida, cuando empiezas esta etapa en la que yo aún soy novel. Son ellos los que te hacen que sea más o menos agradable saltar de la cama la mayor parte de los días, y aquellos que no se den cuenta de que aunque trabajes para vivir tu trabajo es tu vida se están engañando y deberían de cambiar de “work”.

Muchas serán las polémicas con unos y otros, la competitividad va a estar allí por muy “estable” que sea el trabajo, pero hay que saber ser en la medida de lo posible sincero y comprensivo.

Así que en esta parte de la dedicatoria que nadie sabe nunca que poner porque no sabe a quien incluir y a quien no, como fue mi caso, he preferido dejarme de nombres, para evitar malos entendidos, y expresarme con claridad.....

A todos aquellos que devuelven una sonrisa sincera cada mañana, te llaman cuando sienten que les necesitas adelantándose a tu petición de socorro, te cuentan aquello que no sabes sin ánimo prepotente (por ello recurres siempre a ellos) y, aunque en ciertos momentos tengan sus “cosillas” son quienes te dan el empujón que requieres en los momentos de baja forma....

gracias a todos vosotros, que poco a poco os habéis convertido en grandes amigos más que en compañeros en esta choza donde pasamos más tiempo del imprescindible.....

... como dijo alguien a quien conozco muy bien “ a mis amigos y compañeros por las enseñanzas y la amistad que respectivamente me han ofrecido”.

ÍNDICE

VI. PRESUPUESTO

VII. BIBLIOGRAFÍA

I. Resumen.

El proyecto que a continuación se presenta se desarrolla dentro de la línea de investigación llevada a cabo por el Departamento de Electrónica de la Universidad de Alcalá en el campo de la ayuda a la movilidad para personas discapacitadas. El sistema se desarrolla sobre una silla de ruedas, y se pretende que el usuario, proporcionando simplemente los nombres de sala origen y destino a la que desea ir dentro de un entorno estructurado, se desplace de forma cómoda al punto deseado.

En este proyecto se propone una solución robusta para el modelado de entornos cerrados en los que se encuentra al menos cierta estructuración, como ocurre en hospitales, centros sociales, oficinas, etc. Además se presenta como característica el hecho de que el propio edificio contenga la información necesaria para desarrollar el movimiento, lo que permite extrapolar el problema a casi cualquier otro entorno. En cuanto a la generación de trayectorias, se basa en simular las trayectorias que desarrollaría un humano para moverse en este tipo de entornos: curvas simples y líneas rectas, de modo que el movimiento resulte confortable para el usuario.

1. Introducción.

A lo largo de esta memoria se describe el trabajo realizado para el desarrollo de un sistema de navegación autónomo en interiores parcialmente estructurados. La plataforma móvil empleada es una silla de ruedas (ver figura 1.1).

El proyecto se desarrolla dentro de la línea de investigación llevada a cabo por el Departamento de Electrónica en el campo de la ayuda a la movilidad para personas discapacitadas [4] [11] [12] durante más de diez años. Dentro de esta área, el sistema pretende ser un avance en el campo de la navegación autónoma, donde se encuadran trabajos sobre generación automática de trayectorias para robots móviles en entornos más o menos complejos, también desarrollados en el propio departamento [1] [2].

El presente capítulo de introducción pretende exponer, por un lado la organización de la memoria que aquí se introduce, y por otro lado, tanto los objetivos como los fundamentos que ha desembocado en el trabajo que expone.

1.1. Organización de la memoria.

Para facilitar la comprensión del trabajo realizado, la memoria contiene diez capítulos que, se pueden subdividir en tres grandes grupos a la hora de abordar su lectura:

1. *Base del desarrollo*: Constituye la primera parte de la memoria y está compuesta por los capítulos 2 y 3 de la misma. Pretende establecer las bases sobre las que

fundamentar la totalidad de los procesos que permiten desarrollar el sistema de navegación autónoma.

a) En el *capítulo 2* se describe el modelo de la planta sobre la que se desarrolla el proceso de navegación y se realiza un breve recorrido sobre los trabajos previos relacionados de algún modo con el proyecto y a partir de los que éste se ha desarrollado.

b) El *capítulo 3* introduce al lector en la descripción general de los distintos procesos que van a constituir el navegador autónomo, permitiendo observar los objetivos de cada proceso, las variables intercambiadas entre ellos y su organización en el hilo de ejecución.

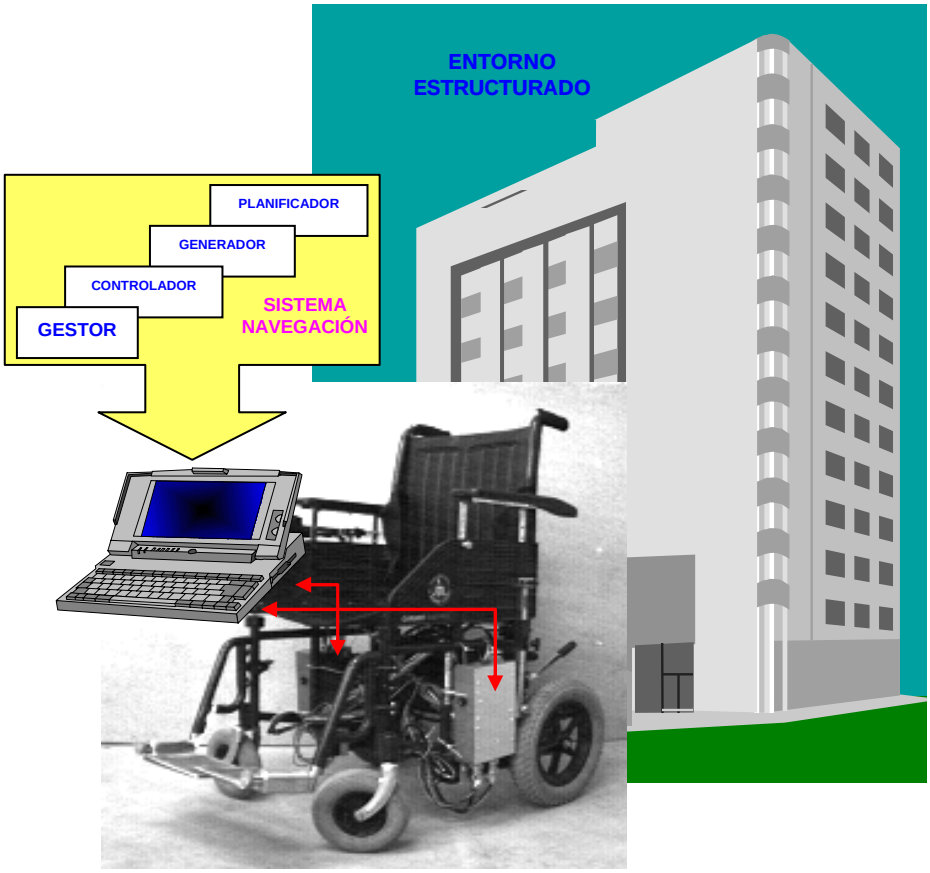


Figura 1.1. Descripción general del proyecto.

2. *Procesos de navegación.* Segunda parte de la memoria, y grueso de la misma. Esta sección está formada por los capítulos 4, 5, 6, 7 y 8 y en ella se realiza la descripción detallada de todo el sistema de navegación.

a) En el *capítulo 4*, y una vez que ya se está familiarizado con el concepto general del proyecto, se realiza una descripción exhaustiva sobre el entorno en el que se centra el trabajo de navegación: La Escuela Politécnica de la

Universidad de Alcalá. En este capítulo se describe cómo diseñar el mapa sobre el que se desarrollan todos los procesos, de modo que el algoritmo de navegación pueda ser transportado a cualquier otro entorno interior y estructurado como el de ejemplo.

b) A lo largo del *resto de capítulos* de este bloque se presentan los distintos algoritmos desarrollados, discutiendo y justificando las distintas opciones adoptadas, y presentado ejemplos de funcionamiento. Al adoptar una filosofía de distribución del sistema de navegación en procesos se ha empleado un capítulo para cada uno de los que forman el proyecto: el gestor de procesos, el planificador de caminos, el generador de trayectorias y el controlador de posición.

3. *Resultados*. Tercer bloque formado por los capítulos 9 y 10, y que presentan las conclusiones del trabajo desarrollado.

   a) En el *capítulo 9* se muestra la exportación del algoritmo completo a la plataforma móvil, realizando una descripción detallada de las tareas de comunicación con el sistema hardware de bajo nivel.

   b) Y, finalmente, en *el último capítulo* de la memoria se exponen los resultados, de los que se obtienen conclusiones y posibles trabajos futuros que quedan planteados para ser desarrollados en la misma línea de investigación.

Para facilitar la lectura, principalmente en los capítulos relacionados con los procesos de navegación, se va a seguir en todos la misma filosofía:

- *En primer lugar* se plantean los objetivos a cumplir por el proceso, así como las premisas de diseño.
- *En segundo lugar* se presentan distintas opciones y se discuten las características de cada una hasta dar con la más adecuada.
- *En tercer lugar* se realiza una descripción detallada del algoritmo elegido, mostrando los resultados de su implementación.

Además de la memoria, en el libro se incluyen otras secciones (pliego de condiciones, presupuesto, y planos, con todos los ficheros fuentes de los algoritmos del navegador) que completan la descripción del desarrollo global del proyecto, así como un manual que permita poner el sistema completo en funcionamiento, y una bibliografía en la que se muestran las referencias utilizadas en el desarrollado de este proyecto.

1.2. Objetivos perseguidos en el trabajo.

Cuando se habla de navegación autónoma muchas veces es necesario concretar a qué nivel el movimiento es autónomo. En este caso, se pretende que el usuario de la silla

de ruedas, proporcionando simplemente los nombres de sala origen y destino a la que desea ir, se desplace de forma cómoda al punto deseado.

Generalmente los sistemas de navegación autónoma desarrollados [10] [13] están aplicados al entorno industrial (carretillas en almacenes, transporte de material dentro de fábricas, transporte de personas por entornos parcialmente conocidos, etc.), pero pocas veces han sido aplicadas al área de la ayuda a disminuidos físicos. En los pocos trabajos desarrollados en este ámbito, se ha probado su gran utilidad en tareas como:

- Transporte de pacientes en hospitales o centros de asistencia.
- Para discapacitados graves, que no pueden emplear métodos convencionales como un joystick para conducir la silla de ruedas.
- Para facilitar el movimiento del usuario por entornos desconocidos para él, de forma que se eviten problemas de accesibilidad o se elijan los caminos más adecuados para conducir la silla de ruedas.

Por otra parte, el trabajo de navegación en este caso y al desarrollarse en interiores, presenta problemas que no aparecen en entornos abiertos o amplios como los industriales: la limitación del espacio, la mayor existencia de obstáculos dinámicos, etc. En este sentido es necesario realizar un esfuerzo importante en el modelado del entorno y la generación de trayectorias adaptadas a estas aplicaciones.

En este proyecto se propone una solución robusta para el modelado de entornos cerrados en los que se encuentra al menos cierta estructuración, como ocurre en hospitales, centros sociales, oficinas, etc. Además se presenta como característica parcialmente innovadora el hecho de que el propio edificio contenga la información necesaria para desarrollar el movimiento, lo que permite extrapolar el problema a casi cualquier otro entorno.

En cuanto a la generación de trayectorias, se basa en simular las trayectorias que desarrollaría un humano para moverse en este tipo de entornos: curvas simples y líneas rectas, de modo que el movimiento resulte confortable para el usuario.

Finalmente, y teniendo en cuenta la misma premisa de comodidad en el movimiento, en el diseño se valora la conducción suave, frente a la eliminación del error en el seguimiento de la trayectoria a costa de fuertes reacciones del controlador.

Con todo ello, el objetivo final del trabajo, es conjugar todas las ideas presentadas para diseñan un sistema que constituya una aportación en la línea de investigación de la navegación autónoma de móviles en entornos cerrados.

2. Trabajos Previos y Modelo de la Silla de Ruedas.

En este capítulo se presentan las bases sobre las que se ideó el presente proyecto en varios aspectos. Por un lado se describen a grandes rasgos los trabajos relacionados con la navegación autónoma, desarrollados fuera y dentro del Departamento de Electrónica. Por otro lado se presenta el modelo del móvil sobre el que se va a realizar el sistema de navegación.

Para ello se realizan continuas llamadas a otros trabajos, cuya referencia completa se incluye en la bibliografía de este documento.

2.1. Trabajos previos.

Muchos son los trabajos llevados a cabo a lo largo de las dos últimas décadas sobre navegación y concretamente en el campo de sistemas de ayuda a la minusvalía. Como ya se comentó en el capitulo 1, en el propio Departamento de Electrónica se han realizado diversos trabajos de desarrollo de una silla de ruedas con distintos módulos de interfaz [16] y movimiento [2] [3] [11] [17] que se adaptan a las características especiales de estos usuarios.

Una posible aportación a los trabajos anteriores es el de la incorporación de un sistema navegación autónoma por interiores estructurados. En este aspecto se ha investigado en trabajos realizados sobre el tema, encontrando referencias e ideas que han podido

ser plasmadas en parte en el trabajo, tal y como se comentará más adelante. Dos de estos trabajos han sido especialmente tenidos en cuenta, quizás por la semejanza de la aplicación [1] [6] y por la proximidad de los desarrolladores.

En este punto se plantean las soluciones propuestas en todos estos casos y el porqué de la elección de algunas frente a otras. Además se presentan soluciones a problemas planteados en dichos trabajos anteriores, y que serán resueltos en este proyecto.

En definitiva se fija la base sobre la que se fundamenta el trabajo presentado, observando por separado cada uno de los módulos que conforman el navegador.

2.1.1. Modelo del entorno.

Por regla general, cualquier trabajo de navegación autónoma requiere en primer lugar un cierto conocimiento sobre el entorno en el que se desarrolla. El formato de la base de datos y la cantidad de información acerca del entorno de movimiento depende de la aplicación concreta, y en los trabajos estudiados el modelo del entorno ha sido distinto.

Existen sistemas de navegación [7] que no tienen ninguna información concreta del entorno, sino que para moverse en el medio en que se encuentran actúan como exploradores del mismo, teniendo, eso si, un cierto conocimiento del tipo de estructuras que conforman el entorno, pasillos de cierta longitud, giros sólo de cierto numero de grados, etc. Se podría decir que estos sistemas son los que presentan una mayor flexibilidad en lo que al medio se refiere, pero no es así, puesto que, como se ha comentado funcionan solo si el entorno presenta una estructura concreta. Además este tipo de navegación presenta la desventaja de que requiere un entrenamiento inicial hasta que el móvil diseña trayectorias optimizadas, durante el cual va diseñando un mapa del entorno que va conociendo.

En estos casos la trayectoria diseñada suele consistir simplemente en ir siguiendo paredes hasta encontrar pasillos, y coger una desviación sin saber si llegará al destino cuando presente proximidad a la trayectoria ideal que seria la línea recta entre origen y destino. Con todo, el sistema ha de poseer al menos de partida la posición del punto de origen y del punto de destino, con lo que como se ve, siempre se parte de cierto conocimiento.

Si bien en el trabajo presentado en el artículo [7] el sistema estaba diseñado para navegación por interiores, es evidente que estos sistemas están especialmente diseñados para tareas de navegación donde el entorno sea totalmente desconocido, sobre todo en trabajos de investigación en fondo oceánico o en el entorno lunar, y requiere un gran soporte sensorial para ir evitando obstáculos, por lo que para la aplicación aquí tratada este sistema no es el más adecuado. Sin embargo hay una idea que merece la pena rescatar de este trabajo, y es el hecho de que estructura el

entorno en dos tipos de trayectorias simplemente: rectas y curvas simples. Esta técnica se va a emplear en muchos otros trabajos [6], y suele facilitar bastante la tarea de codificación del mapa y por tanto la de generación de trayectorias, como se verá más adelante. Se puede sacar como conclusión de este trabajo, que si bien el sistema de navegación ha de ser diseñado para una estructura de entorno determinada y que se requiere cierta información acerca del mismo, no es necesario tener una reconstrucción exacta del entorno de movimiento para el que el robot pueda desenvolverse por él.

En muchos casos, además de contar con un mapa del entorno, el navegador cuenta con un sistema de visión artificial, que mediante algoritmos de reconocimiento e identificación de formas permiten localizar puertas, pasillos y distribuidores en un edificio. Generalmente este elemento se emplea como sistema de realimentación para obtener la posición en un mapa conocido, pues sus limitaciones en cuanto a obstáculos que se interponen en el objetivo, la complejidad de evaluación de la tercera dimensión y la dificultad de identificar todos los elementos que se pueden encontrar en un medio como el estudiado hacen imposible emplearlo de forma genérica en todos los casos. En algunos casos se implementan sistemas de estereovisión o emisiones de luz estructurada (láser) para evitar estos problemas [1].

Una técnica empleada en otro de los trabajos estudiados [8], que parece bastante interesante, es la de utilizar patrones predeterminados que identifiquen ciertos elementos en el entorno de movimiento mediante un sistema de visión. Cuando se busca un patrón conocido, los problemas de los reconocimiento comentados se eliminan, simplificándose además la tarea en gran medida si además estas marcas se encuentran siempre en una misma posición en el entorno (por ejemplo a una determinada distancia del suelo). Dentro de estas técnicas existen trabajos que se basan en lo que llaman "marcas naturales", por tratarse de elementos que se encuentran en el propio entorno sin modificarlo. Sin embargo, comparando los resultados queda bastante claro que estos sistemas presentan una mayor tasa de error en el reconocimiento, además de un mayor tiempo de proceso. Interesa, por tanto, de este trabajo quedarse con la idea del apoyo de las marcas artificiales en el modelado del entorno.

En este mismo trabajo se presenta otra idea que va a ser recogida en este proyecto. Las propias marcas artificiales pueden contener información sobre el medio, con lo que ayudan por sí mismas (sin un mapa anexo) a modelar el entorno. El método consiste en codificar jerárquicamente los distintos puntos que conforman el mapa, de modo que el código que identifica la marca proporciona además información acerca de la localización de ese punto en la estructura jerárquica del medio. Esta táctica es perfecta para el tipo de entornos en los que se desarrolla el proyecto: entornos interiores que siempre presentan cierta jerarquía en su distribución en plantas, pasillos, distribuidores, etc., consiguiendo que finalmente el mapa del entorno lo posea el mismo entorno.

2.1.2. Organización de procesos.

Como ya se ha comentado anteriormente, el navegador ha de ser diseñado de un modo estructurado y con unos objetivos concretos evitando las teorías excesivamente generales sobre navegación. En este aspecto, la organización de procesos juega un papel importante, pues permite diseccionar el problema completo en distintas tareas, que serán afrontadas por el navegador de un modo más sencillo.

En todos los trabajos observados el patrón de organización de procesos es siempre el mismo, aunque la interrelación y el funcionamiento interno de cada uno puede ser distinto.

1. Existe un *planificador de rutas* que encuentra el camino más corto entre el origen y el destino a partir del mapa del entorno, mediante diversos algoritmos.
2. Existe sobre él un *generador de trayectorias*, que permite unir mediante curvas y rectas de diversa índole los puntos que forman la ruta completa, y modifican la ruta si el sistema de sensores detecta un obstáculo en el camino.
3. Finalmente se encuentra el *controlador de posición* que obtiene las consignas de velocidad para el sistema motor que permiten seguir fielmente la trayectoria diseñada por el elemento anterior, a partir de la información de posición real proporcionada por el sistema de posicionamiento absoluto.

El diseño de estos elementos se aborda por separado y para las características concretas del sistema de navegación deseado. La interconexión de los distintos procesos viene dada por un núcleo operativo en tiempo real más o menos complejo, que se encarga de arrancar las tareas y gestionar el intercambio de información entre ellas.

En uno de los trabajos más recientes [6] se emplea un núcleo de Linux en tiempo real para realizar estas tareas. Este sistema facilita la gestión de los procesos pues incorpora herramientas concretas para esta tarea como son: temporizadores con una tasa muy baja de errores de jitter; recursos para compartir información entre procesos con control de coherencia como semáforos, FIFOs, etc.; recursos para asegurar la ejecución de procesos en tiempo real, pues reserva espacio en el núcleo del sistema operativo; un supervisor de las tareas que se encarga de arrancar las distintas tareas en función de su prioridad y permite detectar si se produce algún error en el proceso de ejecución.

Para un correcto uso de estas herramientas, es muy importante definir las características temporales y las necesidades de recursos de cada tarea, siendo solamente interesante su uso en casos donde aparezcan dos o más tareas de ejecución paralela en tiempo real. Es por ello que estos núcleos se suelen emplear en sistemas de navegación con múltiples procesos de sensado y control, como el que se presenta en el trabajo mencionado [6].

2.1.3. Generación de trayectorias.

El proceso de generación de trayectorias, también imprescindible en cualquier trabajo de navegación, es el que realmente diseña el camino que va a seguir el robot en su movimiento. Este proceso es también muy diferente en función del objetivo perseguido por el sistema de navegación, y depende absolutamente del entorno de movimiento en que se centre el trabajo.

Así en uno de los proyectos examinados [1] se realiza un estudio exhaustivo sobre las distintas técnicas empleadas para la planificación de caminos y la generación de trayectorias en el mundo de la navegación. Tal y como se observa en este trabajo, existen diversas teorías sobre obtención de trayectorias que permiten unir puntos mediante curvas de distintos tipos; así se habla de generación de splines, clotoides y otras curvas sobre caminos determinados por un grafo visible, un diagrama de Voronoi o un mapa de campos potenciales.

El uso de unas u otras, de nuevo, depende de la aplicación concreta: el medio, el móvil y los objetivos, siendo la técnica de grafo visible y generación de splines la empleada finalmente en el trabajo mencionado, y la más extendida también. La técnica se divide en dos partes diferenciadas:

- En primer lugar se obtiene el camino como un grafo formado por puntos por los que pasará el móvil para ir al destino deseado. Estos puntos pueden ser celdas desocupadas del mapa o vértices de obstáculos que hay que evitar.

- En segundo lugar, el camino determinado por este grafo se suaviza mediante una consecución de curvas de 2º o 3º orden, que forman lo que será realmente el camino seguido por el móvil.

Este método es válido para cualquier tipo de entorno, por su gran generalidad, y permite fácilmente incorporar cambios de trayectoria para evitar los obstáculos que vayan apareciendo de forma dinámica a lo largo del camino.

Sin embargo para el caso concreto de entornos interiores estructurados es posible reducir el tipo de trayectorias que se van a dar, simplificando el algoritmo de generación de trayectorias y optimizando por tanto las trayectorias específicas que se van a encontrar en estos caminos.

Esta técnica recurre a descomponer el trayecto existente entre cada pareja de puntos del grafo visible en dos tipos de tramos, rectas y curvas simples, imitando el comportamiento humano en la forma de desenvolverse en este tipo de entornos: "avanza más o menos en línea recta", "gira a la derecha 90 grados", etc. Esta técnica que, como ya se ha comentado, se encuentra en varios de los trabajos estudiados [8]

[6], va a ser la empleada en el proyecto por resultar la más adecuada al caso específico que aquí se trata.

2.1.4. Control de posición.

Para poder realizar el movimiento diseñado por el generador de trayectorias es necesario que un sistema de control conozca en cada instante la posición real del móvil y al compararla con la trayectoria deseada obtenga las consignas de movimiento. Es el típico lazo de realimentación que existe en todos los sistemas de control.

El algoritmo de control para conseguir que el movimiento se ajuste lo más posible a la trayectoria diseñada también es muy variopinto en los trabajos estudiados. Como siempre la base está en obtener una serie de consignas sobre las que aplicar una ley de control, lineal o no, que sobre el sistema resulte la más eficaz. Las consignas vendrán establecidas por la ley de control y van desde el simple error de posición del que se obtienen de forma proporcional las consignas de velocidad de giro de las ruedas, a controles más elaborados que minimizan el error en base a un modelo bastante exacto del móvil, o técnicas modernas de control borroso, que permiten incorporar al comportamiento del robot características difícilmente modelables a través de sus parámetros.

Uno de los trabajos más desarrollados sobre el tema [2] permite obtener muy buenos resultados en el seguimiento de trayectorias mediante controladores óptimos y borrosos, pero requiere una plataforma hardware específica y bastante rápida que permita ejecutar los complejos algoritmos de control. En el proyecto presentado en esta memoria, se ha preferido emplear controladores más sencillos que, incorporando un error aceptable a la trayectoria a seguir, no supongan el grueso de la aplicación concreta de navegación sino un proceso más, al tratarse de un proyecto menos ambicioso a nivel de control.

Si bien el algoritmo de control que se emplea en los distintos trabajos estudiados difiere en cada caso, en la mayor parte de los proyectos realizados el sistema de posicionamiento del móvil se basa en el deadreckoning, es decir, en la recuperación de la posición a partir del movimiento de las ruedas del móvil (generalmente mediante tacómetros ópticos de tipo encoder). Este método no es quizás el más adecuado, tal y como se explicara más adelante, debido al carácter no holonómico de su cinemática, pero es el más recurrido por tratarse del más inmediato y sencillo.

Las limitaciones del sistema de deadreckoning se ven paliadas en la mayor parte de los sistemas bajo estudio por otro elemento de posicionamiento absoluto basado en la mayor parte de los casos en sistemas de visión artificial o de ultrasonidos. Estos sistemas por si solos tampoco permitirían realizar el posicionamiento absoluto, unas

veces por su lentitud de procesado y otras por limitaciones intrínsecas al sistema (la tercera dimensión en la visión y los falsos blancos de los ultrasonidos).

2.2. El modelo de la silla de ruedas.

Para poder implementar un cierto control del movimiento del robot sobre el que se realiza el sistema de navegación, va a ser necesario tener un modelo matemático del mismo.

Este modelo va a ser necesario no solo para analizar el control de posición que se diseña sino también para poder obtener la posición real del móvil si se desean emplear técnicas de deadreckoning.

En varios de los trabajos analizados previamente [2] se presenta el modelo de la silla de ruedas con las siguientes entradas y salidas (ver figura 2.1).

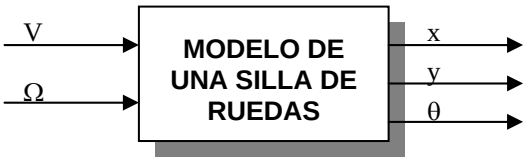


Figura 2.1. Diagrama de bloques del modelo de una silla de ruedas.

Tal y como se aprecia en la figura, las entradas del modelo son *la velocidad de avance (V)* y *la velocidad de giro (Ω)* de la silla.

a) Teniendo en cuenta que la dirección de avance de la silla de ruedas es siempre la del eje longitudinal de la silla, y que el centro de giro de cualquier móvil se encuentra en la intersección de las perpendiculares a todas sus ruedas, se conoce como *velocidad de avance de la silla de ruedas (V)* a la velocidad con que se mueve el punto de intersección de estos dos ejes, tal y como se aprecia en la figura 2.2.

b) *La velocidad de giro de la silla (Ω)* indica, por otro lado, la variación de la orientación de la silla en un incremento de tiempo.

Por otro lado las variables de salida del modelo elegidas informan de la posición de la silla en respuesta a estas consignas de velocidad. Esta posición vendrá definida por tres variables (ver figura 2.2):

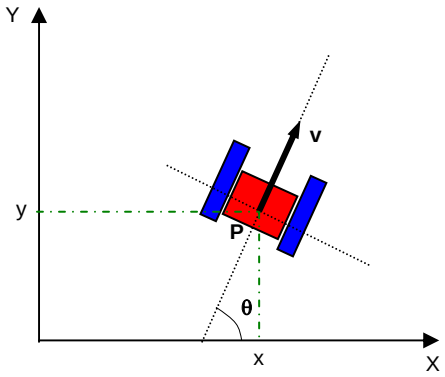


Figura 2.2. Diagrama explicativo de las variables de entrada y salida del modelo de una silla de ruedas.

a) Por un lado el par (x,y) informa de las coordenadas cartesianas del centro de movimiento de la silla, en un sistema de coordenadas definido en el espacio en el que se desenvuelve la silla.

b) Por otro lado, la orientación (θ) informa del ángulo formado por el eje de movimiento de la silla (eje longitudinal) y el semieje positivo de abscisas del sistema de coordenadas elegido, en sentido antihorario.

A partir de la definición de las entradas y salidas, es inmediato obtener las ecuaciones del modelo de la silla [2], que serán las siguientes (<2.1>, <2.2> y <2.3>):

$$\dot{\theta} = \Omega \qquad [rad/s] \qquad \text{<2.1>}$$

$$\dot{x} = V[n]\cdot\cos\theta \qquad [m/s] \qquad \text{<2.2>}$$

$$\dot{y} = V[n]\cdot\sin\theta \qquad [m/s] \qquad \text{<2.3>}$$

Por otro lado, si bien las consignas obtenidas por el controlador de posición en la mayor parte de los sistema de este tipo son V y Ω, tal y como se ha comentado, los controladores de la silla de ruedas requieren como consignas, las velocidades angulares de cada una de las dos ruedas motrices (ω_D para la rueda derecha y ω_I para la rueda izquierda).

Las ecuaciones que describen la relación de las consignas con estas nuevas variables son lo que se conoce como cinemática de la silla, y pueden ser obtenidas fácilmente a través de los parámetros de la silla, tal y como se muestra en la figura 2.3 [2]. Son las ecuaciones <2.4> y <2.5>.

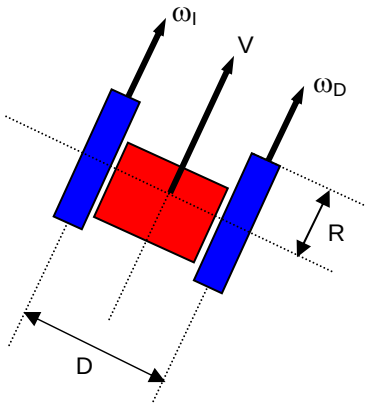


Figura 2.3. Diagrama explicativo de las relaciones cinemáticas de la silla de ruedas.

$$\omega_D = \frac{1}{R}\cdot\left(V + \Omega\frac{D}{2}\right) \qquad [rad/s] \qquad <2.4>$$

$$\omega_I = \frac{1}{R}\cdot\left(V - \Omega\frac{D}{2}\right) \qquad [rad/s] \qquad <2.5>$$

Las relaciones inversas (V y Ω en función de ω_D y ω_I) constituyen lo que se viene denominando como cinemática inversa.

A la hora de trabajar con el móvil, el sistema que realmente se va a presentar es el que se muestra en la figura 2.4. Al emplear un sistema de realimentación de posición en base a encoders ópticos incrementales, las salidas que proporciona la silla de ruedas son las velocidades angulares de las ruedas (ω_D y ω_I), a partir de las cuales el sistema de deadreckoning ha de obtener las variables de posición (x, y, θ). Para ello se basa en integrar la posición recorrida por la silla en cada periodo de ejecución del algoritmo de control.

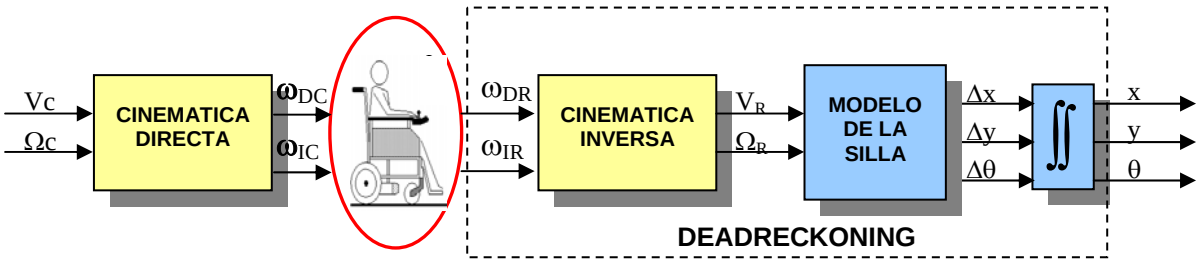


Figura 2.4. Diagrama explicativo del sistema real de realimentación.
Leyenda de subíndices:
X_C: Consigna.
X_R: Medida real.

En el proceso de deadreckoning se puede obtener la ubicación real de la silla de ruedas en cada instante de muestreo, conociendo la posición anterior de la misma (x[n-1], y[n-1], θ[n-1]) y mediante el modelo discretizado de ésta, tal y como se muestra a continuación (<2.6>, <2.7>, <2.8>).

$$\theta[n] = \theta[n-1] + \Omega[n] \qquad [rad] \qquad \text{<2.6>}$$

$$x[n] = x[n-1] + V[n] \cdot \cos(\theta[n]) \cdot Ts \qquad [m] \qquad \text{<2.7>}$$

$$y[n] = y[n-1] + V[n] \cdot sen(\theta[n]) \cdot Ts \qquad [m] \qquad \text{<2.8>}$$

Es muy importante el papel del periodo de muestreo, pues para que el modelo discreto responda del mismo modo que el continuo, este debe ser al menos el doble de rápido que la respuesta de la planta. En uno de los trabajos ya mencionados [5], se realiza un estudio sobre el límite de dicho tiempo, observando que hasta los 200ms, las diferencias entre las respuestas del modelo continua y discreto son inapreciables. Este aspecto deberá ser tenido en cuenta a la hora de fijar el periodo de ejecución del algoritmo de control, tal y como se verá en el capítulo 8.

Además queda en este punto de manifiesto el problema que presenta el uso de un posicionador absoluto como el de deadreckoning en estos casos. Debido a su constitución, la silla de ruedas se comporta como un sistema no holonómico, pues el estudio independiente de cada una de las partes móviles no basta para obtener el movimiento absoluto del sistema. Es por ello que para conocer la posición real del móvil es necesario integrar, con lo que los errores de posicionamiento cometidos en el proceso de integración puede introducir errores de posición graves, pues son acumulativos.

Debido a esto en muchos casos se añaden estimadores que, a partir de la observación del movimiento de las ruedas, permitan obtener la posición real del sistema sin errores. Uno de los métodos más empleados es el Filtro de Kalman [15]. En este proyecto tampoco se ha incluido un elemento de este tipo, por lo que en el controlador diseñado se tendrá especial cuidado en evitar aceleraciones bruscas, que son las situaciones más propensas para que se cometa un error de integración en el proceso de deadreckoning.

3. Descripción Global del Sistema.

Observadas, evaluadas y criticadas las soluciones que se adoptan en otros casos ante problemas de navegación como el que aquí se plantea, el sistema completo que se va a desarrollar es finalmente y a grandes rasgos el que se muestra en la figura 3.1.

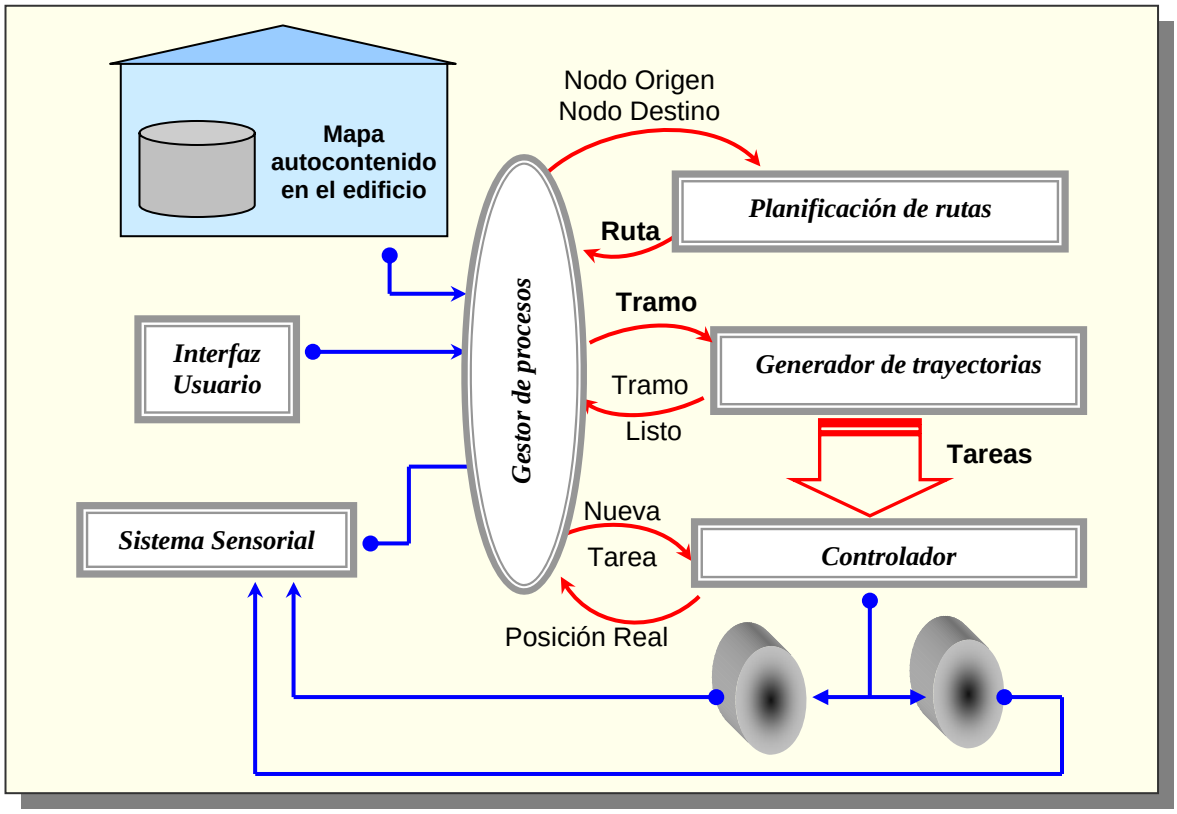


Figura 3.1. Diagrama funcional del sistema de navegación.

3.1. Introducción.

El modo de organizar las tareas se basa en la idea del navegador basado en la subdivisión jerárquica del problema en procesos más pequeños, ya presentado en el capítulo 2 de la memoria. En la figura 3.1 se muestra claramente dicha subdivisión, así como la forma de organizar los procesos para implementar el sistema de navegación deseado.

Tal y como se muestra en dicha figura, los procesos básicos del sistema de navegación son:

- El *gestor de procesos*, encargado de sincronizar el resto de procesos. Está supervisando el conjunto de procesos jerárquicos, y se podría decir que actúa de sistema operativo del conjunto, pues a través de él se realizan las llamadas a los procesos necesario, se gestionan las comunicaciones, etc.

- *El planificador de rutas*, que obtiene el camino más corto entre el origen y el destino indicados por el usuario. Para llevar a cabo esta tarea se emplean algoritmos basados en diagramas de nodos.

- *El generador de trayectorias*, que diseña la trayectoria que ha de seguir el móvil a lo largo del camino.

- *El controlador de posición*, que se encarga de transformar las consignas de posición en actuaciones de velocidad para enviar a las ruedas del robot, y de conseguir que la trayectoria se siga con un error tolerable.

Como sistema jerárquico de procesos, los distintos algoritmos se encuentran organizados en forma de capas, de modo que cada uno proporciona servicios al nivel de la capa superior, y, conforme se localizan en capas más internas, menor es el grado de abstracción con el que trabajan con los parámetros de movimiento. En la figura 3.2 se observa un diagrama de capas del sistema bajo estudio.

En esta figura se puede ver como el controlador es el nivel que ocupa la capa más profunda, pues está directamente en contacto con el bajo nivel de la silla de ruedas, mientras que el planificador de trayectorias es el elemento que trabaja a un nivel más abstracto (a nivel de sala o habitaciones), encontrándose por tanto en la capa más alta de la jerarquía.

Finalmente, también en la figura 3.2, se observa que el proceso de gestión de trayectorias se encuentra en contacto con todas las capas, pues de este modo se pretende señalar que tiene acceso a todos los procesos en su función de organizador de tareas.

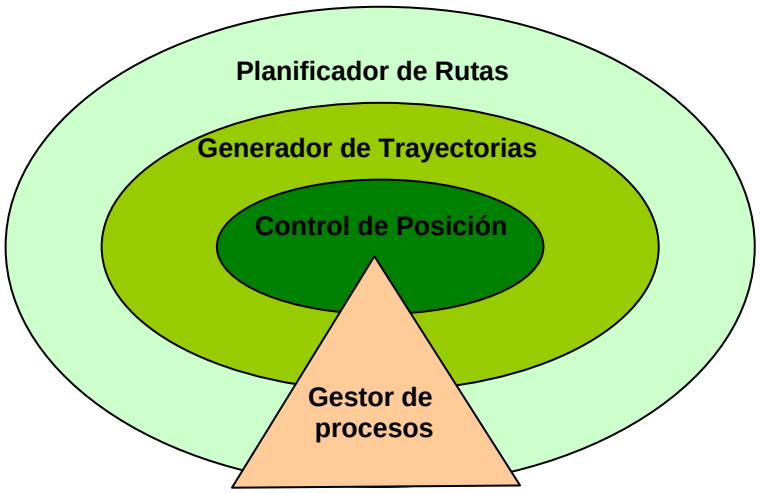


Figura 3.2. Organización en capas de los procesos básicos del sistema de navegación

A parte de los procesos básicos, es necesario que existan otros elementos para que el movimiento se lleve a cabo. Estos son:

- *El sistema de bajo nivel,* con los motores y sus tarjetas excitadoras.
- *La interfaz de usuario*, a través del cual se introduce el objetivo del proceso de navegación.
- *Un mapa del entorno,* necesario para poder desarrollar la tarea de navegación.
- *Un sistema de posicionamiento absoluto,* que permite al controlador de posición conocer la posición real del robot para, comparándola con la consigna, actuar sobre los motores del móvil.

En la figura 3.1 se han presentado también las distintas consignas de entrada y salida de cada uno de los módulos que conforman el sistema global, ayudando así a describir la funcionalidad y el nivel de abstracción de cada uno de los elementos. Es interesante observar tres de ellas que, como ya se comentaba, permiten subdividir el problema global de la navegación en partes más pequeñas. Estos elementos son:

a) *La ruta,* lista de nodos que forman la trayectoria para ir del origen a destino pedidos por el usuario

b) *Los tramos,* que resultan de subdividir la ruta en parejas de nodos que son origen y destino parcial del camino completo.

c) *Las tareas,* que informan sobre las consignas de posición y condiciones de movimiento para completar las distintas trayectorias que forman un tramo.

Tal y como se ha comentado con anterioridad, el proyecto se centra en la implementación sobre un sistema real de todos los procesos básicos que constituyen un sistema de navegación autónomo, para ponerlo a prueba sobre una silla de ruedas

moviéndose en un entorno estructurado como es el Edificio Politécnico de la Universidad de Alcalá.

En este capítulo se incluye una breve explicación sobre la plataforma construida y su funcionamiento a grandes rasgos, para, en capítulos posteriores, pasar a desarrollar una explicación más detallada.

3.2. Descripción general de los procesos básicos de navegación.

En los distintos capítulos de la memoria se realiza una descripción detallada de cada uno de los procesos presentados en la figura 3.1, y que constituyen el motor del sistema de navegación.

Para poder comprender dicha descripción es necesario tener una idea general del procedimiento completo, por lo que en este apartado se pretende realizar una descripción somera del funcionamiento de cada uno de ellos.

3.2.1. Gestor de Procesos.

Como ya se ha comentado, el gestor de procesos tiene una función de organizador y supervisor de las tareas que se desarrollan en el sistema de navegación global. Como tal, en un proceso estándar de movimiento de la silla de ruedas entre un punto de origen y otro de destino, el gestor de procesos realizaría las siguientes tareas:

1. Recoge de la interfaz de usuario las consignas de punto de inicio y fin.
2. Realiza la llamada al planificador de caminos y recoge la ruta que proporciona como resultado.
3. Divide el recorrido global en tramos y realiza las llamadas pertinentes al generador de trayectorias para obtener las tareas correspondientes a cada tramo.
4. Supervisa la ejecución del algoritmo de control cada periodo de muestreo, así como la evolución real de la silla de ruedas.

Debido al trabajo que en este nivel se lleva a cabo, el gestor requiere a veces procesar información relacionada con la posición deseada (a través del mapa de entorno) y con la posición real (a través del sistema sensorial de posicionamiento), deadreckoning en principio).

Todos los aspectos referentes al gestor de procesos serán abordados en el capítulo 5 de la memoria.

3.2.2. Planificador de Rutas.

El primer módulo es el planificador de trayectorias. Este elemento recibe las consignas directamente de la interfaz de usuario, de modo que trabaja a un nivel de abstracción elevado: origen y destino de la tarea final de movimiento (*"ir de la cocina al salón"*). Como salida, este procesador ha de proporcionar la consigna para el siguiente sistema en la jerarquía, que será *la ruta*.

El hecho de trabajar mediante diagramas de nodos, unido al modelado jerárquico del entorno que se ha desarrollado, facilita mucho este proceso, tal y como se verá más adelante, en el capítulo 6 de esta memoria.

El planificador de rutas necesita información del entorno para trazar la ruta que une el destino con la posición actual del móvil. Esa información es proporcionada por el mapa del entorno, y, tal y como se observa en la figura 3.1, está autocontenida en el edificio y presenta una distribución jerárquica en forma de nodos. Además, el planificador de rutas ha de estar comunicado, evidentemente, con el módulo de interfaz de usuario, cosa que ocurre indirectamente a través del gestor de procesos.

3.2.3. Generador de Trayectorias.

El nivel intermedio en la jerarquía de procesos lo constituye el generador de trayectorias, que trabaja ya con datos de posición con lo que el nivel de abstracción es menor que en el caso anterior.

Este elemento recibe del gestor de procesos un tramo que ha de subdividir en tareas y obtener para cada una de ellas las consignas que necesita el controlador de posición para describir la trayectoria asociada a cada una de las tareas. Para unir los dos nodos que conforman un tramo, el generador diseña dos tipos de tareas: *de avance y de giro*, correspondiéndose respectivamente la primera con una trayectoria en forma de línea recta y la segunda con un segmento de circunferencia.

Tal y como se verá en detalle en el capítulo 7 de la memoria, la elección de estos dos tipos de trayectorias está basada en proporcionar al usuario una mayor comodidad y naturalidad en la conducción del móvil de forma autónoma.

Para realizar su cometido, el generador de trayectorias tiene acceso directo al mapa del entorno, que le permite determinar en qué casos ha de incorporar una tarea de avance y en cuales una de giro.

Los resultados obtenidos de la generación de trayectorias han de pasar al controlador, para lo cual se emplea un sistema de buffers controlados por flags de sincronismo que controla, evidentemente, el gestor de procesos.

3.2.4. Controlador de Posición.

Finalmente, en el nivel inferior de abstracción queda el controlador, que trabaja directamente sobre el control de bajo nivel del móvil. Se trata de un controlador de posición que, en función de la información de posición real del móvil y del tipo de trayectoria que se desea describir, obtiene las consignas directas para proporcionar directamente al bajo nivel la velocidad angular que han de seguir de las ruedas del robot.

Para poder llevar a cabo el control, este último módulo requiere de un sistema de realimentación de la posición, que le permita comparar la posición consigna con la real del móvil y obtener a partir del error de posición las consignas de velocidad. Tal y como se describe en el capítulo 8 de la memoria, en el proyecto se ha empleado como único sistema de posicionamiento el sistema de deadreckoning de la silla, basado en encoders ópticos acoplados directamente al eje de los motores DC de las ruedas.

Sin embargo, todo el sistema de navegación ha sido pensado para incorporar un sistema de posicionamiento absoluto que, además, queda perfectamente descrito a nivel funcional en la memoria, y que no ha sido implementado por encontrarse fuera de las expectativas finales del proyecto.

3.3. Los otros procesos del sistema de navegación.

Existen otros procesos que también tienen su parte importante en el sistema global de navegación y que, bien por no haber sido desarrollados en profundidad, bien por ser secundarios en el algoritmo de navegación autónoma en sí, se reúnen en este punto.

3.3.1. El bajo nivel.

Evidentemente, para poder desarrollar los algoritmos de navegación ha sido necesario emplear una silla de ruedas perfectamente acondicionada.

Para ello se ha empleado una plataforma que cuenta con dos motores de continua excitados por sendos puentes en H, que, a su vez, son controlados por microcontroladores programados al efecto. Todo el desarrollo del bajo nivel se llevó a cabo en el proyecto "Guiado Semiautomático de una Silla de Ruedas" ([5]), si bien varios trabajos anteriores [3] [4] [11] ya habían desarrollado versiones anteriores.

Como se puede observar el trabajo realizado por el Departamento de Electrónica en este tema se remonta ya casi diez años, por lo que la plataforma de trabajo se encuentra perfectamente probada.

Como controladores de bajo nivel se emplea el integrado *Neuron Chip 3120* de Echelon, sistema orientado a la implementación de redes de control distribuido [19]. Empleando esta característica, la plataforma consta de una red de control distribuido basado en el protocolo LonWorks (EIA709), desarrollado por Echelon e implementado en sus niveles OSI más bajos en el propio Neuron Chip, haciendo el trabajo transparente al usuario.

Los integrados de Echelon llevan además sobre los motores de continua de la silla un control PI de velocidad, que permite regular perfectamente la consigna de velocidad enviada por el navegador de alto nivel.

Para implementar la comunicación entre el navegador y el bajo nivel se ha empleado una tarjeta de interfaz que permite adaptar el protocolo EPP del puerto paralelo por el que se envían los comandos al protocolo LonWorks existente en el bajo nivel.

Dicha tarjeta, desarrollada en un trabajo previo ([18]), cuenta con una memoria Dual Port que permite comunicar los dos elementos consiguiendo una velocidad de transferencia de alrededor de 1Mbps, por lo que no constituye un cuello de botella en el sistema de navegación.

En la figura 3.3 se muestra un esquema físico del sistema completo, en el que se puede apreciar la localización de cada uno de los procesos del navegador y el bajo nivel y el uso de la tarjeta de interfaz.

El funcionamiento del sistema de interfaz, así como la explicación detallada del algoritmo de control de bajo nivel serán abordados en el capítulo 9 de la memoria.

3.3.2. La interfaz de usuario.

Tal y como se muestra en la figura 3.1, es imprescindible un elemento que permita comunicar al sistema de navegación el destino elegido por el usuario. Esta es la tarea más importante del sistema de interfaz de usuario.

De hecho, en este proyecto es la única tarea desarrollada una interfaz de usuario, si bien este campo da mucho juego en cuanto a la información que sería posible intercambiar entre el usuario y el navegador, como se irá viendo a lo largo de la memoria.

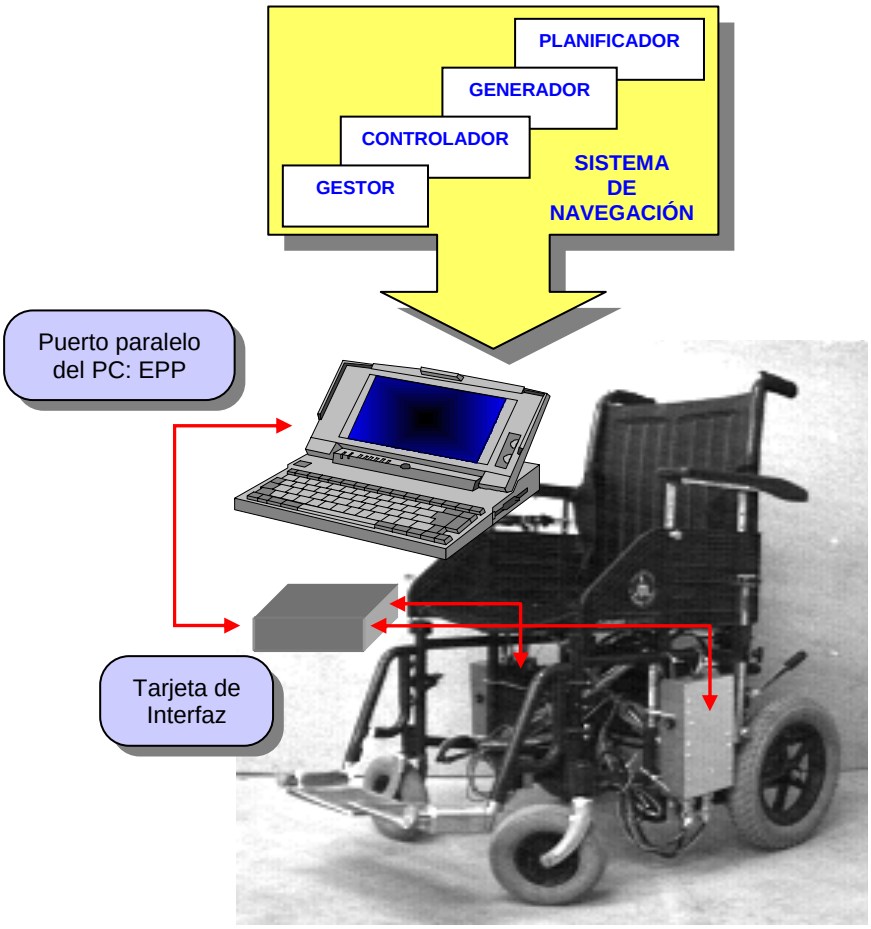


Figura 3.3. Diagrama físico del sistema de navegación.

3.3.3. El modelado del entorno.

Tal y como se ha visto en capítulos anteriores, el modelado del entorno en que se va a desenvolver la silla de ruedas es imprescindible para desarrollar los procesos de navegación

Es por ello que se dedica el capítulo 4 a realizar un estudio exhaustivo sobre el Edificio Politécnico de la Universidad de Alcalá, por ser el medio empleado en el proyecto como ejemplo de entorno estructurado por el que se mueve la silla.

De este estudio se extrae el mapa de entorno que van a emplear los procesos de planificación de rutas, generación de trayectorias y control de posición, para conseguir que el móvil alcance el destino esperado. Es necesario establecer unos criterios que permitan incluir en el mapa la información física y topológica acerca de los planos del edificio bajo estudio, necesaria para desarrollar los procesos de navegación.

3.3.4. El sistema de visión.

A lo largo de toda la exposición se realizan numerosas referencias a un sistema de posicionamiento absoluto que complemente a la plataforma completa desarrollada, tal y como se muestra en la figura 3.1.

Tras realizar un estudio, a lo largo del desarrollo del proyecto y en colaboración con otros trabajos [12], se resuelve que el elemento más adecuado para desarrollar esta tarea es un sistema de visión artificial. Las tareas para las que se requiere este proceso son, fundamentalmente, las siguientes:

1. Detectar y procesar las marcas artificiales que ayudan a la localización jerárquica y física de la plataforma.

2. Corregir los errores de posicionamiento que se puedan incorporar con el sistema de deadreckoning.

3. Implementar un sistema de detección de obstáculos que ayuden al sistema de generación de trayectorias a retrazar el camino evitando estos obstáculos.

4. Permitir incorporar una tarea de localización de marcas cuando la plataforma se encuentre en un estado de desorientación

De las tres tareas comentadas, es la primera la única que ha sido expuesta con detalle en la memoria. Las otras dos funciones requieren un estudio específico mucho más profundo y serían tema de proyectos de investigación, pues en los textos manejados no se propone una solución robusta para todos los casos.

Estos y otros temas, que se plantean pero a los que no se da una solución en el proyecto se proponen al final de la memoria como trabajo a desarrollar en el futuro, sirviendo sin embargo el presente proyecto como iniciación en un tema tan puntero como el de la navegación autónoma.

4. Modelado del Entorno

El modelado del entorno consiste en la interpretación del entorno de movimiento mediante figuras geométricas que constituirán el mapa del entorno. Esta tarea constituye una de las partes más importantes en los algoritmos de navegación, pues permitirá al resto de los procesos de navegación tener la información que necesita para llevar a cabo el movimiento.

Existen numerosas técnicas de modelado, tal como se puede apreciar en el capítulo 2. Sin embargo, el proceso de generación del mapa es siempre específico para el entorno de que se trate, por lo que finalmente, no existen más que ciertos algoritmos básicos a partir de los que se construye el modelo de la aplicación concreta [7].

En este caso, el diseño del mapa se adapta al medio de navegación de esta aplicación concreta: entornos cerrados parcialmente estructurados. Como ya se comentaba el entorno elegido para realizar las pruebas de campo es, concretamente, el Edificio Politécnico de la Universidad de Alcalá.

Sin embargo el trabajo ha sido desarrollado con suficiente generalidad como para ser adaptable a cualquier otro entorno de las mismas características sin más que obtener el mapa adecuadamente organizado del medio donde se desea emplear.

4.1. Consideraciones sobre el entorno de movimiento.

La técnica de modelado elegida debe permitir obtener un mapa de entorno adecuada al medio en que se desarrolle la navegación. Así el mapa será distinto en función de que el móvil se vaya a desenvolver en un medio estático o dinámico, interior o exterior, conocido o desconocido, estructurado o aleatorio, etc.

Los algoritmos de modelado también serán muy distintos según el tipo de entorno bajo estudio, así en medios desconocidos o dinámicos será necesario una tarea de reconocimiento y estructuración on-line del medio, mientras que en medios estáticos conocidos, el mapa se construye off-line, con lo que no es necesario perder tiempo de ejecución en este punto.

Si además el medio es interior y estructurado, como es el caso, el mapa puede incluso facilitar la tarea del resto de procesos de navegación (planificación de caminos, generación de trayectorias, etc.), pues puede incorporar información extra que permita hacer más cómodos los movimientos del robot. Este es un aspecto que se desea tener en cuenta en el proyecto, pues no es lo mismo diseñar un sistema de navegación para una carretilla industrial que para una silla de ruedas, en la que viajará una persona.

La desventaja que introduce un mapa estático es la de que, una vez diseñado, el robot solo va a poder moverse en el entorno determinado, o en caso de cambiar de medio es necesario realizar una carga off-line del nuevo mapa. En este proyecto se ha deseado también mejorar este aspecto, haciendo que el mapa se encuentre autocontenido en el edificio por el que se realice la navegación. De este modo, la silla de ruedas al acceder a un nuevo entorno realiza de forma automática la carga del mapa asociado. Además, en caso de que existan cambios temporales en el mapa (un corte en un pasillo o ascensor sin funcionamiento), no es necesario provocar una nueva carga del mismo, pues de forma automática al acceder al mismo los nuevos datos serán cargados. En apartados posteriores se profundizará en esta idea.

Finalmente parece interesante destacar el hecho de que la mayoría de los modelados se realizan para entornos estáticos parcialmente dinámicos, es decir, que existe la posibilidad de que aparezcan obstáculos que se interpongan en el camino y sea necesario, por tanto, modificar la trayectoria inicial del móvil.

Este tratamiento mixto del entorno es, quizás, el más adecuado para la aplicación presentada en el proyecto. Para poder incorporar esta posibilidad, el robot ha de incluir un sistema sensorial que permita detectar y caracterizar adecuadamente al obstáculo. Como el diseño de este sistema se sale de las expectativas del proyecto, el sistema de navegación se diseña para entornos totalmente estáticos, sin incluir la posibilidad de evitar obstáculos, al menos de forma activa. En el capítulo final de la memoria (capítulo 10) se aborda este tema con mayor profundidad.

4.2. Método de modelado del entorno.

Una vez se tiene el entorno perfectamente determinado por medio de un mapa estático, es necesario realizar la descripción del algoritmo de modelado elegido, adaptándolo, tal y como ya se ha mencionado, al caso particular de la aplicación bajo estudio.

4.2.1. Organización nodal del entorno.

Todos los métodos conocidos de modelado de entorno se basan en la reducción del mapa a un *grafo* [1], sobre el que se aplicará un algoritmo de planificación de caminos.

Un grafo se puede considerar un diagrama formado por líneas, que se llaman ramas y que sirven para interconectar puntos, llamados nodos. La mayor parte de los mapas se limitan a establecer los nodos que se encuentran en el entorno, definidos a través de variables de posición. En muchos casos, los nodos determinan el camino a seguir, pues forman parte del camino, estos modelos son los que a partir de ahora se van a llamar *nodales*.

Sin embargo estos modelos no son los únicos. De hecho un estudio más profundo sobre los métodos empleados para obtener el grafo de configuración, resultaría en distinguir los dos grandes grupos que tradicionalmente se emplean:

1. *Métodos de proyección,* consistentes en dividir el espacio de configuración en celdas, representando cada una de ellas un nodo. En este caso la ruta está formada por una secuencia de los nodos que se encuentran entre punto origen y de fin, que han de coincidir también con sendos nodos. Son, por tanto, modelos nodales.

2. *Métodos de retracción,* basados en este caso en reducir el espacio de configuración empleando como elementos delimitadores los vértices de los obstáculos del mapa (técnicas de grafo visible) o algún tipo de esqueleto del mapa (diagramas de Voronoi o campos potenciales). En este caso el camino no ha de pasar por los nodos, por lo que no se considerarán métodos nodales.

En este proyecto se emplea una técnica de tipo proyectivo, pues son las más adecuadas para entornos completamente estáticos. En este caso los nodos del mapa se asignan según las premisas siguientes:

- Empleando las características de estructuración propias de edificios como el empleado de ejemplo, cada nodo se asocia a una sala, un acceso a un pasillo o un cruce entre pasillos.

- Los nodos son además *jerárquicos*, pues, mediante métodos de codificación adecuados, se relacionan entre ellos formando una estructura piramidal, que facilita la planificación de rutas.

Por tanto, la forma de modelar el entorno para poder diseñar las rutas que permitan unir dos salas cualquiera del entorno será mediante un *mapa jerárquico de nodos.* Además de facilitar la tarea de planificación, el hecho de que el mapa sea jerárquico permite realizar fácilmente una identificación lógica de la posición real del móvil, tal y como se verá en posteriores apartados.

4.2.2. Identificación física del mapa nodal.

Habitualmente la planificación de caminos de mapas de tipo nodal se basa en nodos virtuales, sin implementación física alguna, pues son simplemente pares de coordenadas *x,y* en el mapa. En este caso los nodos cumplen una única función: la del entorno.

Los nodos del modelo de mapa implementado en el proyecto no son nodos virtuales, sino que tienen una representación física concreta: existen unas marcas colocadas en el edificio (*landmarks*) en ciertas posiciones estratégicas de las salas del edificio (generalmente el en quicio superior de las puertas de acceso a la sala o al pasillo). Cada landmark identifica al nodo asociado en el mapa topológico.

De esta forma, es posible establecer una estrecha relación entre el mapa topológico (nodal) y el mapa físico, que posibilita el hecho de que los nodos jerárquicos del modelo empleado en el proyecto cubran dos funciones:

- Por un lado cumplen la tarea de *modelar el entorno* de trabajo, tal y como ocurre con los nodos virtuales tradicionales.
- Por otro lado sirven, además, para realizar un *posicionamiento absoluto* del móvil mediante sistemas de visión.

Como sistema de posicionamiento absoluto, las marcas son elementos que sirven para corregir la posición real del móvil en el mapa, ya que el posicionamiento mediante deadreckoning provoca muchos errores debido a deslizamientos [1] [2]. Utilizando las landmarks como nodos se realiza una realimentación y corrección de la posición en cada nodo justo antes de afrontar el nuevo tramo, y/o , al estar colocadas en el punto de acceso a la sala, para facilitar el paso del móvil a través de las puertas, lo cual constituye una de las trayectorias más complicadas de la navegación autónoma.

La asociación del mapa autocontenido y las marcas visibles constituye una de las características más atractivas de la técnica de modelado presentada. A través de las marcas el edificio, incorpora la información necesaria para la navegación, con lo que se recurre de nuevo a la idea de que el móvil no tiene por qué almacenar toda la

información sobre el edificio en que se mueve sino que el propio edificio las proporcionas cuando es necesario.

Como se comentó en el capítulo 2, la idea de emplear elementos físicos que ayuden a localizar el móvil en el mapa topológico ya se ha puesto en práctica de distintas formas en varias ocasiones en trabajos previos. Sin embargo el uso de las marcas en su función dual es bastante novedoso.

La colocación de marcas artificiales ya ha sido ampliamente explotada en diversos trabajos de navegación [9]. De hecho, un tema de gran actualidad es el uso de marcas naturales, como ventanas, cuadros, u otros elementos propios de interiores de edificios, que se puedan usar como elementos de localización absoluta. Este método, sin embargo, no ha dado buenos resultados, pues es sensible a los cambios de decoración del entorno, además de que estos elementos naturales presentan una mayor dificultad de ser identificados por los algoritmos de visión.

El método estándar de nodos virtuales tiene también sus ventajas frente a los nodos físicos, que sin embargo no son suficientes como para contrarrestar las del uso de nodos visibles:

- El uso de nodos virtuales elimina el engorro de tener que colocar marcas artificiales en el edificio de interés para localizar los nodos. Sin embargo, en cualquier caso sería necesario incluir algún otro elemento de posicionamiento absoluto artificial (p.ej. balizas) y si se desea emplear un métodos de localización por visión artificial (métodos que, además, son los empleados en la mayor parte de los casos de navegación por su mayor robustez y simplicidad) el método más robusto es el de emplear marcas artificiales, tal y como se ha discutido anteriormente.

- Además, el uso de nodos virtuales elimina (o al menos disminuye) la dependencia del sistema de navegación respecto del de visión (en cuanto a fiabilidad y velocidad de proceso). Pero de nuevo aparecerá esta dependencia al incluir éste u otro sistema sensorial para las tareas de posicionamiento absoluto, que no se incluyen entre las funciones de los nodos virtuales (recordar la dualidad de estos nodos físicos).

En cualquier caso, se ha de decir que el sistema desarrollado en este proyecto es fácilmente adaptable al caso de nodos sin landmarks, ya que de hecho, en este proyecto no se cuenta con el apoyo de un sistema de visión que permita identificar estas marcas.

4.3. Identificación topológica del mapa de nodos.

El algoritmo modelado de entorno basado en mapas jerarquizados permite estructurar fácilmente el entorno a tratar. De este modo, la tarea de planificación de rutas es una tarea sencilla que consiste en recorrer un árbol jerárquico siempre con las mismas pautas, tal y como se describirá en el capítulo 6 de la memoria.

De la idea anterior se desprende que es necesario establecer un sistema de codificación jerárquico de nodos antes de pasar a realizar la asignación pertinente, ya que el objetivo es que de este código jerárquico se obtenga información sobre la posición real del nodo.

Para conseguir incorporar esta información a los nodos, cada uno de ellos se identifica fundamentalmente por dos elementos: *el número y el nombre de nodo:*

1. *El número de nodo* es el código numérico que aparece representado en la marca física asociada al nodo y colocada por tanto en la sala o cruce del edificio correspondiente. Su única función es la identificar a la marca frente a las otras, por lo que no proporciona ninguna información sobre la localización de la marca en el entorno. Se asocia, por tanto, a la parte física del nodo.

2. *El nombre del nodo* identifica al nodo en cuanto a la posición jerárquica que ocupa en el mapa (información topológica) y es la información que se va a emplear para conocer la localización del nodo en el árbol jerárquico que constituye el mapa. Esta información va a ser por sí sola suficiente para realizar la planificación de la ruta en algunos casos. Por todo esto, el nombre se asocia a la parte topológica del nodo.

De este modo, el planificador de rutas genera una lista de nombres de nodos, que se asocian a los números de nodos que identifican en definitiva a las habitaciones por las que va pasando el móvil en la ruta, a la vez que permiten realizar un reajuste en el sistema de posicionamiento del móvil.

En los puntos siguientes de este capítulo se describe cómo se ha realizado realmente la asignación del número y nombre para cada uno de los nodos que constituyen el mapa del entorno base de esta aplicación.

4.3.1. Organización física del mapa. Número de Nodo.

Como ya se ha comentado, el número de nodo identifica físicamente al nodo en el edificio, al ser el código que aparece en las marcas artificiales colocadas en el entorno.

Si bien el sistema de visión artificial para detectar y reconocer las marcas no ha sido desarrollado en el proyecto, se ha realizado un pequeño estudio acerca de la codificación y funcionalidad de las landmarks, apoyando el trabajo en un proyecto de tesis paralelo [12].

En este estudio, la marca artificial se construye mediante la unión de dos partes diferenciadas (ver figura 4.1) :

1. Cada marca consta de un patrón de localización que no aporta información para identificarla sino solo para reconocerla mediante el sistema de visión adecuado. Se trata de un patrón sencillo pero que permite discriminar fácilmente la marca en un medio interior como el estudiado.
2. Por otro lado tiene un campo que presenta el número de nodo asociado a la marca mediante un sistema de codificación determinado.

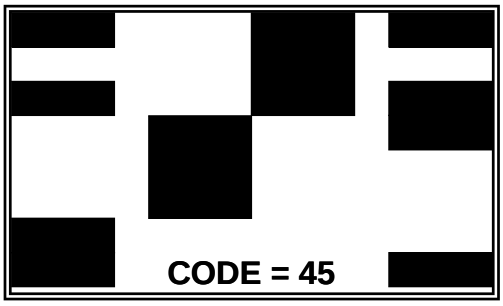


Figura 4.1. Ejemplo de marca artificial para el modelado físico del entorno.

Para numerar las marcas se elige como sistema de numeración, en primera instancia, un código de barras por su sencillez de procesado a la par que robustez en la detección. El sistema de codificación elegido permite implementar dos series de números: A0-A99 y B0-B99. Con esta codificación se pueden distinguir hasta un total de 200 marcas, lo que significa que el mapa puede tener hasta un total de 200 nodos.

Como se verá más adelante, esta cifra es suficiente para codificar la totalidad de los nodos en la aplicación bajo estudio, sin embargo, si eso no fuese así, no hay que perder de vista la idea de que el propio edificio contiene el mapa que lo identifica, y que será transferido al sistema de navegación de forma on-line cuando sea necesario. Esta característica se puede aprovechar tanto para recorrer varios edificios con el mismo algoritmo de navegación, como para dividir el edificio en distintos mapas topológicos y cargar uno u otro en función de la localización del móvil en cada instante.

Esta utilidad es empleada en el sistema bajo estudio, pues el edificio bajo estudio cuenta con una distribución por alas que se va a aprovechar para dividir el mismo en distintos mapas. En la figura 4.2 se muestra la distribución de mapas dentro del Edificio Politécnico sobre el plano de la tercera planta del mismo.

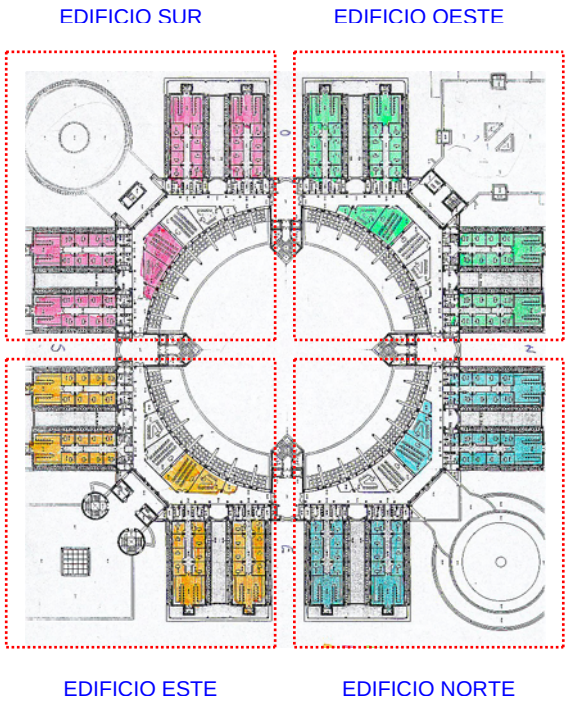


Figura 4.2. Distribución de mapas de la tercera planta del Edificio Politécnico.

Lo que se conoce como mapa en este trabajo no es más que un fichero ASCII que cuenta con un listado de nodos, para cada uno de los cuales se incorpora cierta información relacionada con su identificación, posición relativa en el entorno y otros parámetros que permiten realizar un diseño de trayectorias cómodo, adecuado para el usuario y el entorno en que se mueve.

Por tanto, cuanto menor sea el número nodos del mapa, la información que se necesita en cada instante será más fácilmente manejable por los algoritmos de planificación, generación y control de trayectorias, dando mayor velocidad de ejecución al proceso completo.

La función del número de nodo es importante dentro del mapa, ya que en éste los nodos se ordenar en función de su número de nodo, que se emplea, de este modo, como índice en la tabla *look-up* que constituye el mapa, facilitando de nuevo el tratamiento del mismo. En posteriores apartados se observa con más detenimiento la construcción del mapa y la función del número de nodo dentro de él.

Para simplificar el software de búsqueda de un nodo en el mapa, el número de nodo se transforma con respecto a su valor en las marcas artificiales, de forma que en vez de ser un código alfanumérico se convierta en uno totalmente numérico simplemente sustituyendo la A por un 0 y la B por un 1, de forma que las dos series que se codifican con el código de barras indexan nodos del número 0 al 99 y del 100 al 199.

Es importante tener en cuenta que la marca identifica la sala o acceso en que se encuentra el nodo, además de su posición. Es por ello que existirá solamente una marca por acceso, y se deberá colocar en un punto por el que el móvil tenga que pasar necesariamente para así facilitar la detección. Por ello se eligen los puntos de acceso o las puertas de las salas como lugar idóneo para su colocación, localizándose además a la altura necesaria como para que se interfiera difícilmente en su visualización (ver figura 4.3).

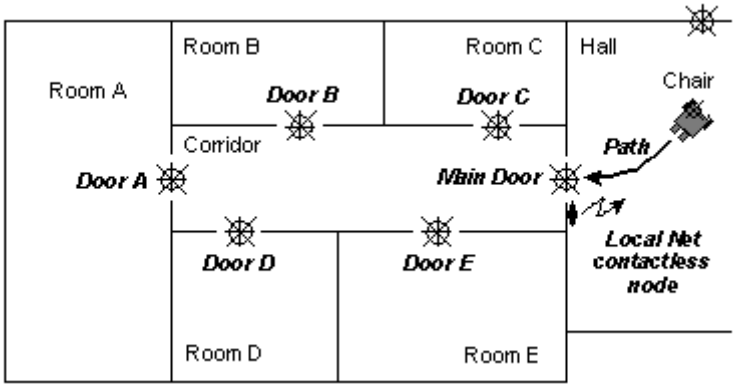


Figura 4.3. Localización de marcas artificiales en el entorno de trabajo.

4.3.2. Organización topológica del mapa. Nombre de nodo.

El nombre del nodo es el primer campo de la información asociada a cada nodo que aparece en el mapa de entorno. Este campo identifica al nodo del mismo modo que lo hace el número, pero además incorpora información sobre la localización del nodo, tal y como ya se ha comentado.

La idea para modelar el mapa con los nombres de nodo se basa en emplear la organización jerárquica de los distintos espacios que se encuentran en un entorno estructurado como el de trabajo.

Todos los entornos de interiores presentan esta característica de jerarquización: en todo encontramos accesos a pasillos y distribuidores, alas o secciones distintas, así como una distribución en plantas. La idea está en utilizar estos elementos para codificar el nombre del nodo y saber así donde se encuentra, simplemente analizando su nombre.

En el caso concreto que se plantea, los elementos de decisión serán el *ala, pasillo y planta* en el que se encuentre el nodo. De este modo es posible generar prácticamente toda la ruta entre cualquier pareja de salas del Edificio Politécnico, simplemente conociendo el nombre de los nodos de origen y destino, sin que la existencia de un

mapa, en muchos casos, sea necesaria para realizar la planificación de rutas completa: el mapa queda implícito en la estructuración del entorno.

Teniendo en cuenta el planteamiento previo y el entorno de trabajo para el que se ha diseñado el algoritmo (3 niveles de jerarquía), la asignación del nombre de nodo a cada sala o punto de acceso se realiza mediante un código alfanumérico de tres cifras. Cada dígito de éste caracteriza a un nivel jerárquico del edificio, de modo que existen tres niveles de jerarquía en la distribución del mapa en cuestión.

- *Nivel 1º (más alto en cuanto a jerarquía):* Identifica a la planta del edificio en que se encuentra el nodo. Es el dígito más significativo en el nombre del nodo.
- *Nivel 2º:* Identifica al pasillo o al acceso o cruce de pasillos en que se encuentra la sala.
- *Nivel 3º (más bajo en cuanto a jerarquía):* Determina la sala concreta asociada al nodo. Es el dígito menos significativo en el nombre del nodo.

Con todo ello la organización jerárquica del entorno se muestra en el diagrama piramidal de la figura 4.4.

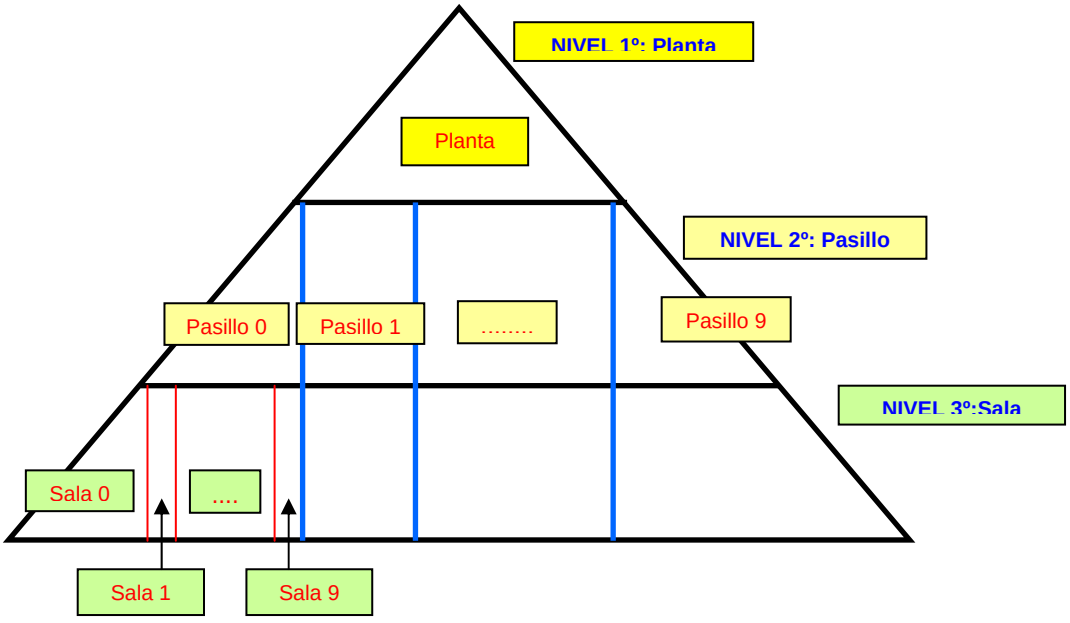


Figura 4.4. Diagrama piramidal de la organización jerárquica del mapa.

De este modo, el nombre localiza perfectamente a cada nodo (pasillo, distribuidor, despacho, aula o laboratorio, ascensor, biblioteca, cafetería, etc.) dentro del mapa. El hecho de que se asigne a cada nivel jerárquico una cifra del código ASCII permite incorporar hasta 127 elementos dentro de un mismo nivel jerárquico. En este caso se ha empleado simplemente una codificación numérica, de modo que se pueden implementar hasta 10 elementos por cada nivel jerárquico, que, como se verá a continuación es suficiente para esta aplicación. Sin embargo, para otro caso se puede

emplear todo el rango ASCII para codificar cada nivel, conservando siempre la característica comentada.

En la figura 4.5 se muestra de forma gráfica el modo de construir el nombre de un nodo, teniendo en cuenta el árbol jerárquico mostrado en la figura 4.4.

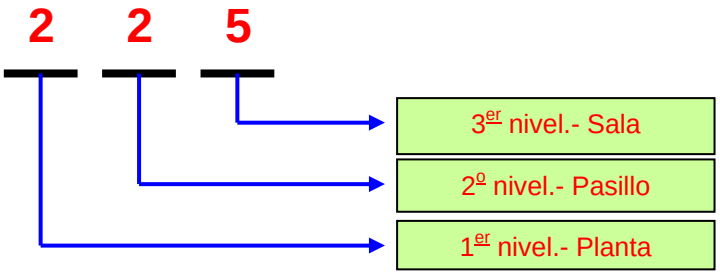


Figura 4.5. Modelo gráfico de formación del nombre de los nodos.

El Edificio Politécnico se distribuye además en distintos mapas, tal y como ya se ha comentado. Esta información no aparece, evidentemente, en el nombre de los nodos del mapa, pues el nombre del nodo se referirá en cada instante al mapa que esté cargado en el móvil. Si el usuario quiere dirigirse a un punto que se encuentra en otro mapa distinto al de origen solamente el planificador de rutas tiene conocimiento de ello, y, como se verá posteriormente en el capítulo 6, diseña el camino adecuado para recoger la información del mapa de destino, de modo que el hecho sea transparente para el resto de procesos del navegador.

Para identificar el mapa en que se encuentra el punto de destino seleccionado, el interfaz de usuario incorpora al nombre del mapa una cifra más. Es lo que se conoce como *nivel 0 del mapa.*

De este modo, en un edificio tan estructurado como el que se estudia, la codificación de nombres se vuelve a repetir en cada ala (norte, sur, este, oeste) distinguiéndose solo por el código del nivel 0, que indica el ala donde se encuentra el nodo (ver figura 4.6).

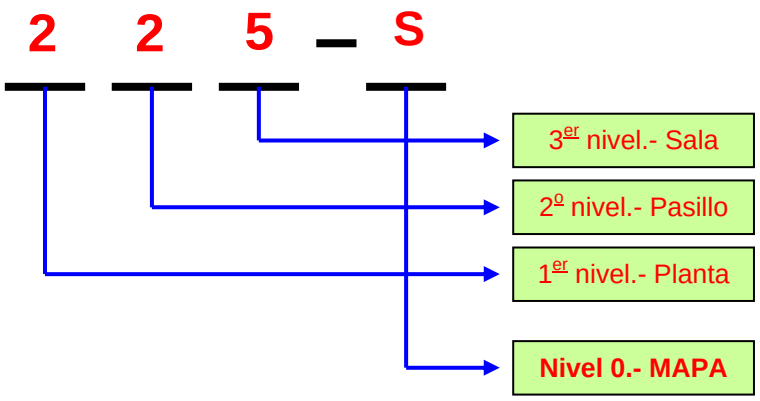


Figura 4.6. Modelo gráfico de formación del nombre de los nodos, incluyendo el nivel 0.

El algoritmo de modelado presentado está muy orientado al entorno donde se va a mover el móvil en esta aplicación concreta, pero es igualmente válido para todos aquellos edificios que presenten un alto nivel de estructuración. Generalmente esto ocurre en casi todos los edificios de interés público para los que está pensada la aplicación: facultades, colegios, hospitales y otros tipos de centros sociales, si bien es quizás algo más complicado en otros entornos como casas particulares, donde no existe una distribución jerárquica tan clara.

Finalmente, en la figura 4.7 se muestra un ejemplo de como queda el mapa del edificio, con el nombre de distintos nodos.

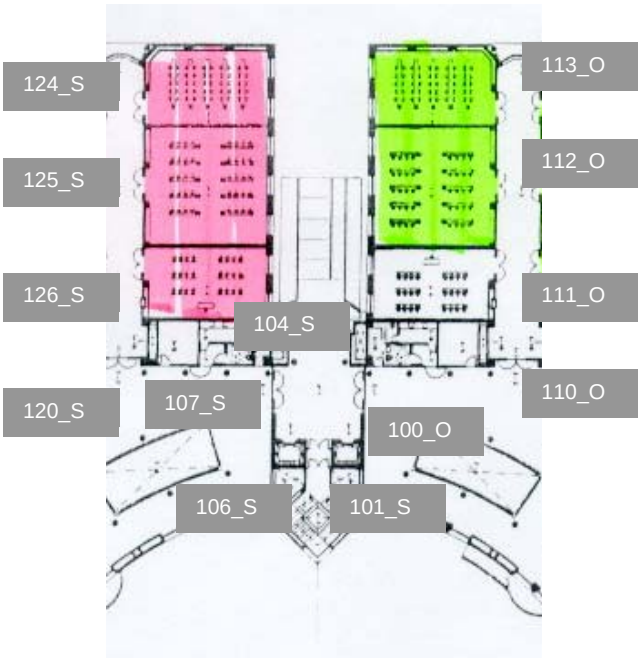


Figura 4.7. Ejemplo de asignación de nombres a nodos en el plano del mapa de la aplicación.

4.4. Tipos de nodos.

De forma paralela a la distribución jerárquica de los nodos, mostrada en la figura 4.4, se puede observar una clasificación de los nodos en función del tipo de estancia a la que identifican en el entorno bajo estudio. Con este parámetro de clasificación aparecen tres clases distintas de nodo en cualquier entorno estructurado como el de la aplicación. Se presentarán en primer lugar dos de los tres tipos:

a) El caso más general es el de nodos que representan a las distintas salas del entorno, y que serán generalmente el origen y destino de las rutas diseñadas por el planificador. Son lo que se llamarán *nodos sala.*

b) Los *nodos de jerarquía* constituyen el segundo tipo. Son nodos incorporados por el planificador de rutas cuando es necesario realizar un cambio en alguno de los niveles jerárquicos para ir del nodo de origen al de destino y, por contra, no van a ser, generalmente, ni origen ni destino de una ruta. En el sistema de codificación de nombres elegidos se caracterizan por que su nombre siempre acaba en 0. Los accesos a pasillos o alas del edificio y los ascensores son ejemplos de nodos jerárquicos en cualquier entorno estructurado.

En la mayor parte de los sistemas de navegación diseñados la organización nodal del mapa del entorno se basa en dividir el recorrido completo del móvil en tramos que incluyan un único cambio de dirección. De este modo se simplifica el trabajo el generador de trayectorias, que ha de trazar la línea más adecuada que una los dos nodos y que evite la colisión con los obstáculos estáticos y dinámicos del entorno. Este es el trasfondo general de cualquier sistema de planificación de caminos, y su correspondiente generación de trayectorias, (mapa basado en nodos y generador de splines, mapa basado en esqueleto, etc.).

Sin embargo, tal y como se ha planteado hasta ahora el concepto de nodo, éstos pierden el sentido indicado, ya que no representan necesariamente un cambio de dirección sino que se emplean para poder abordar más fácilmente el movimiento en un entorno estructurado. Es por ello que en algunos casos va a ser necesario incorporar nodos sin significado jerárquico sino simplemente de trayectoria. Son el tercer tipo de nodos, los *nodos de transición*.

c) Los nodos de transición son nodos que, como los jerárquicos, tampoco constituyen origen ni destino de una ruta normal, pero, al contrario que estos, no son origen de una jerarquía dentro del árbol del entorno. Estos nodos se incluyen para facilitar la generación de trayectorias, informando de que en torno a ese punto es necesario realizar un cambio de dirección en la misma.

Los nodos de transición son nodos virtuales, en el sentido de que no existe una marca artificial relacionada con ellos al no identificar ningún punto de interés a nivel jerarquía del mapa. Debido a esta faceta han de ser incorporados al final del fichero que contiene el mapa del entorno, pues de otro modo se perdería la función de puntero del número de nodo, tal y como se explica en el apartado 4.2.1.

Además, debido a su particular significado, al nombre de los nodos de transición se les añade como prefijo el caracter ASCII '-'.

En la figura 4.8 se muestra de nuevo un segmento del plano de la tercera planta del ala oeste del Edificio Politécnico, donde aparecen representados todos los nodos del mapa en distintos colores según el tipo: en rojo los de transición, en verde los de jerarquía y en malva los de sala.

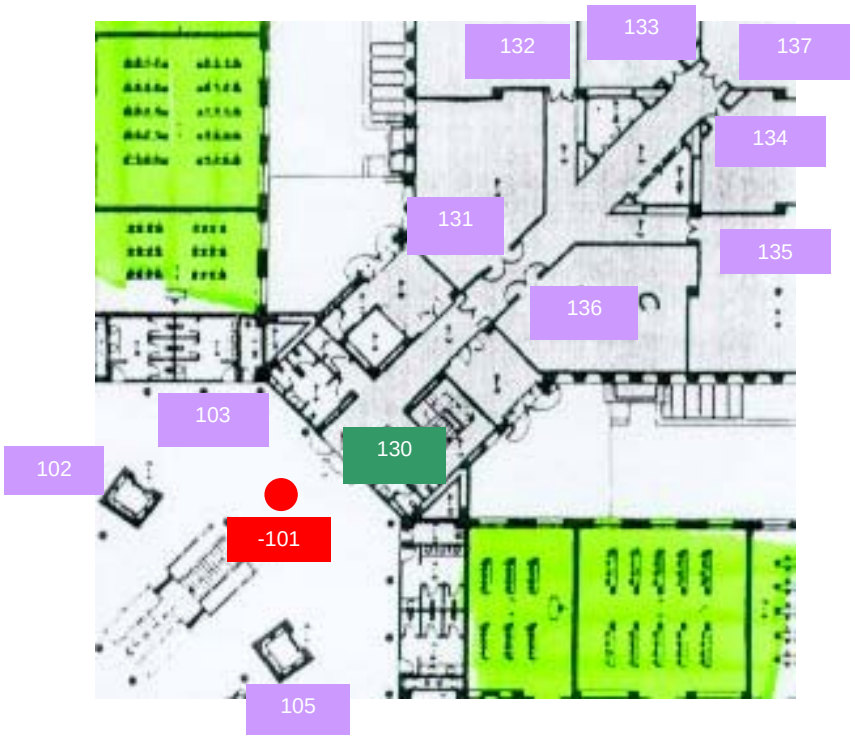


Figura 4.8. Plano de ejemplo de localización de nodos con información sobre tipología.
Malva: Nodos de sala
Verde: Nodos de jerarquía
Rojos: Nodos de transición

4.5. Mapa del entorno.

El edificio bajo estudio está dividido en cuatro alas de distribución muy semejante, y las pruebas se han centrado en el edificio oeste (ver figura 4.2), por ser el caso intermedio de dificultad en cuanto a distribución y la actual ubicación del Departamento de Electrónica.

Es necesario realizar un estudio profundo de este entorno que permita obtener el mapa del mismo a partir de las premisas presentadas a lo largo de todo el capítulo.

El edificio cuenta con 4 plantas, dentro de las cuales se distinguen las siguientes habitaciones:

Planta baja: Donde se encuentran las aulas y algunas de las zonas comunes.

- 2 pasillos con 4 aulas cada uno.
- 1 baño para minusválidos.
- 1 pasillo central que da acceso a 3 laboratorios de investigación.
- 1 puerta de acceso al patio central del edificio.

- 2 ascensores en la zona central y 2 en los distribuidores que comunican con las alas norte y sur.
- 2 puertas de acceso a los propios distribuidores comunicantes con las alas norte y sur.

Planta 1º: Donde están los laboratorios docentes y más salas de uso común.

- 2 pasillos con 6 laboratorios docentes cada uno.
- 1 baño para minusválidos.
- 1 pasillo central que da acceso a 7 salas comunes, donde se encuentran entre otras la sala de reprografía y la de delegación.
- 2 ascensores en la zona central y 2 en los distribuidores que comunican con las alas norte y sur.
- 2 puertas de acceso desde la zona central del ala a los distribuidores que la comunican con las alas norte y sur.

Planta 2º: Donde aparecen parte de los despachos de los profesores y más laboratorios comunes.

- 4 pasillos de despachos con 8 despachos y una sala común al fondo.
- 2 baños para minusválidos.
- 1 pasillo central que da acceso a 7 laboratorios de investigación.
- 2 ascensores en la zona central y 2 en los distribuidores que comunican con las alas norte y sur.
- 2 puertas de acceso a los distribuidores que la comunican con las alas norte y sur.

Planta 3º: Donde se encuentran el resto de despachos de profesores y la zona común del departamento.

- 4 pasillos de despachos con 8 despachos y una sala común al fondo.
- 2 baños para minusválidos.
- 4 salas comunes del departamento, con la secretaría la sala de reuniones, la de documentación y la fotocopiadora.
- 2 ascensores en los distribuidores que comunican con las alas norte y sur.
- 2 puertas de acceso a los distribuidores que la comunican con las alas norte y sur.

Cada una de las salas enumeradas tiene su nodo correspondiente, sumando un total de 157 nodos, número posible de codificar con el sistema de numeración de marcas elegido. Además aparecen varios nodos de transición (que no necesitan número de nodo):

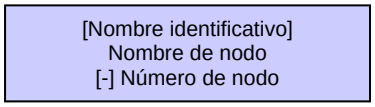
- *Planta baja:* 1 en la zona central.
- *Planta 1º:* 1 en la zona central y 1 en el pasillo central de zonas comunes.

- *Planta 2º:* 1 en la zona central y 1 en el pasillo central de zonas comunes.
- *Planta 3º:* 2 en el pasillo central.

Con todo ello se construyen las figuras 4.9, 4.10, 4.11 y 4.12 en las que se muestra el árbol jerárquico completo de los nodos encontrados en el ala oeste del edificio. El resto de los mapas sería semejante en cuanto a numeración y distribución de nodos, cambiando solamente la distribución de las zonas comunes en algunos casos.

Cabe destacar que la secuencia de numeración de los nodos no se ha hecho siguiendo un criterio concreto, sino de forma aleatoria.

En los organigramas se usa la siguiente leyenda:



Además se sigue la codificación de colores que se indica a continuación:

- *Nodos Azules:* Son los nodos de sala, que, como ya se ha comentado, se corresponden con una sala concreta del mapa
- *Nodos Verdes:* Son nodos de jerarquía. Se observa claramente que no dan acceso a ninguna sala concreta, sino a un pasillo o distribuidor.
- *Nodos Rojos:* Son los nodos de transición. No presentan el campo de número de nodo, pues, como se ha comentado, son nodos virtuales (no tienen marca artificial asociada), empleados para facilitar la generación de trayectorias.

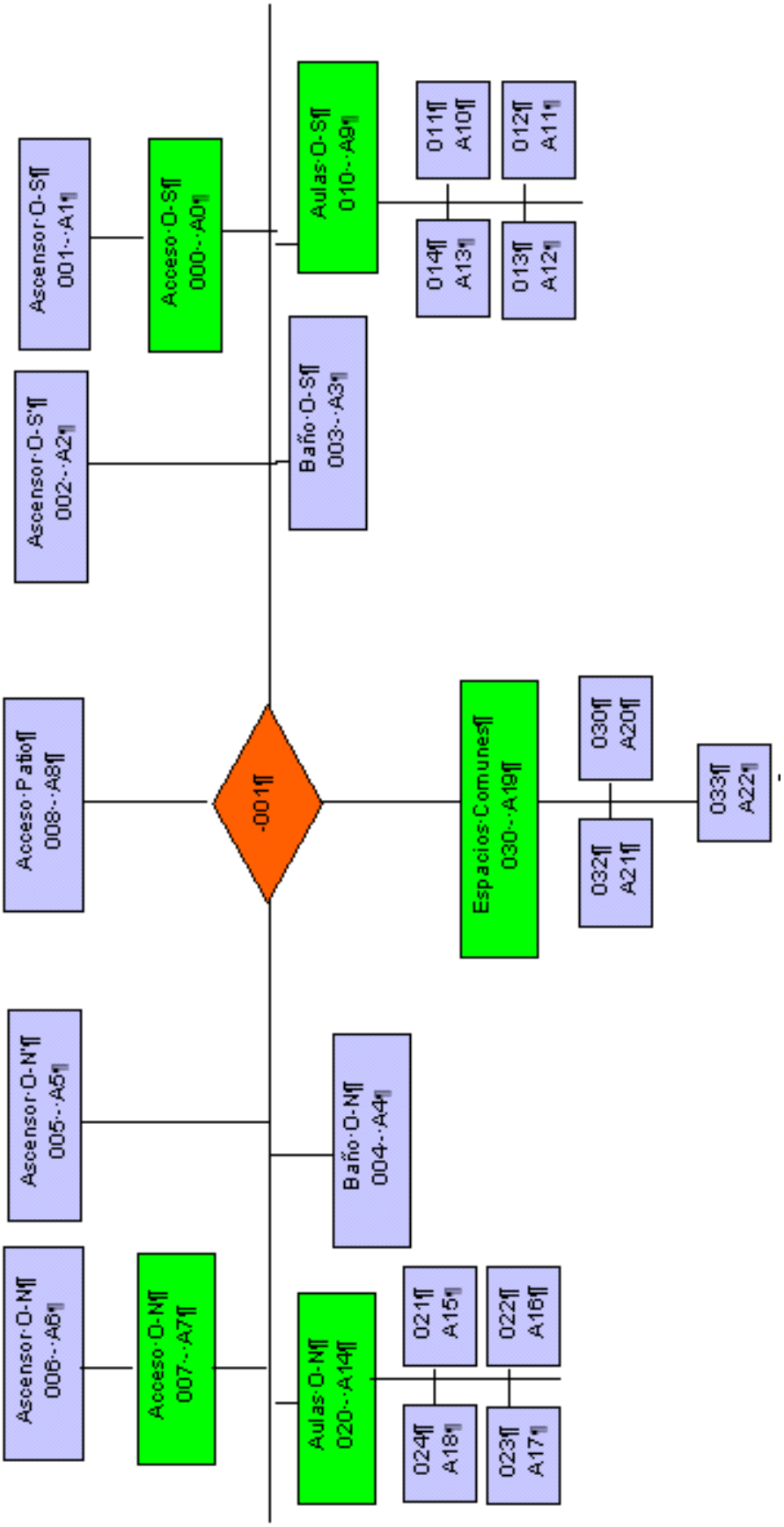


Figura 4.9. Arbol jerárquico de nodos en la planta baja del edificio oeste de la Escuela Politécnica

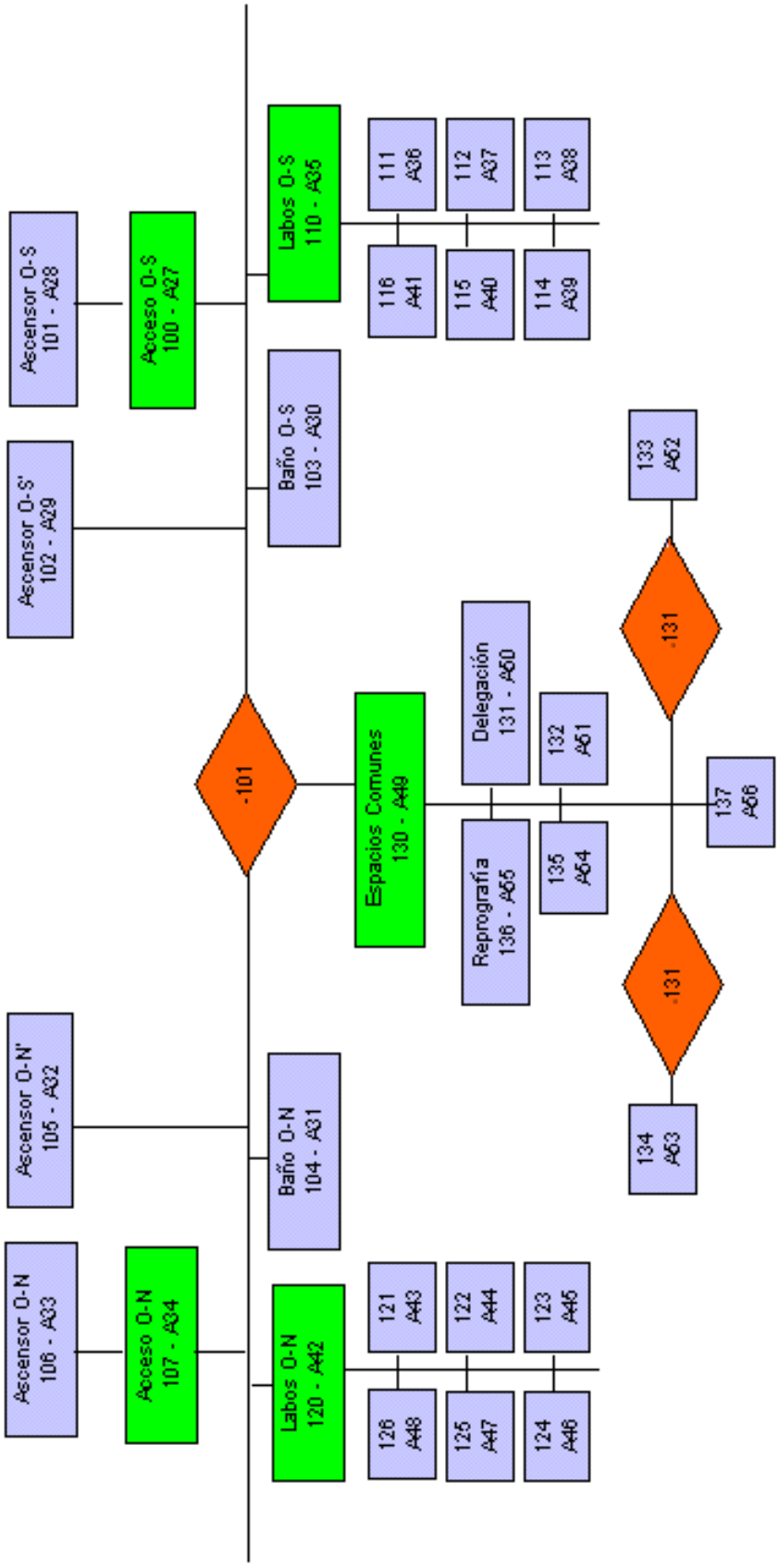


Figura 4.10. Arbol jerárquico de nodos en la planta primera del edificio oeste de la Escuela Politécnica

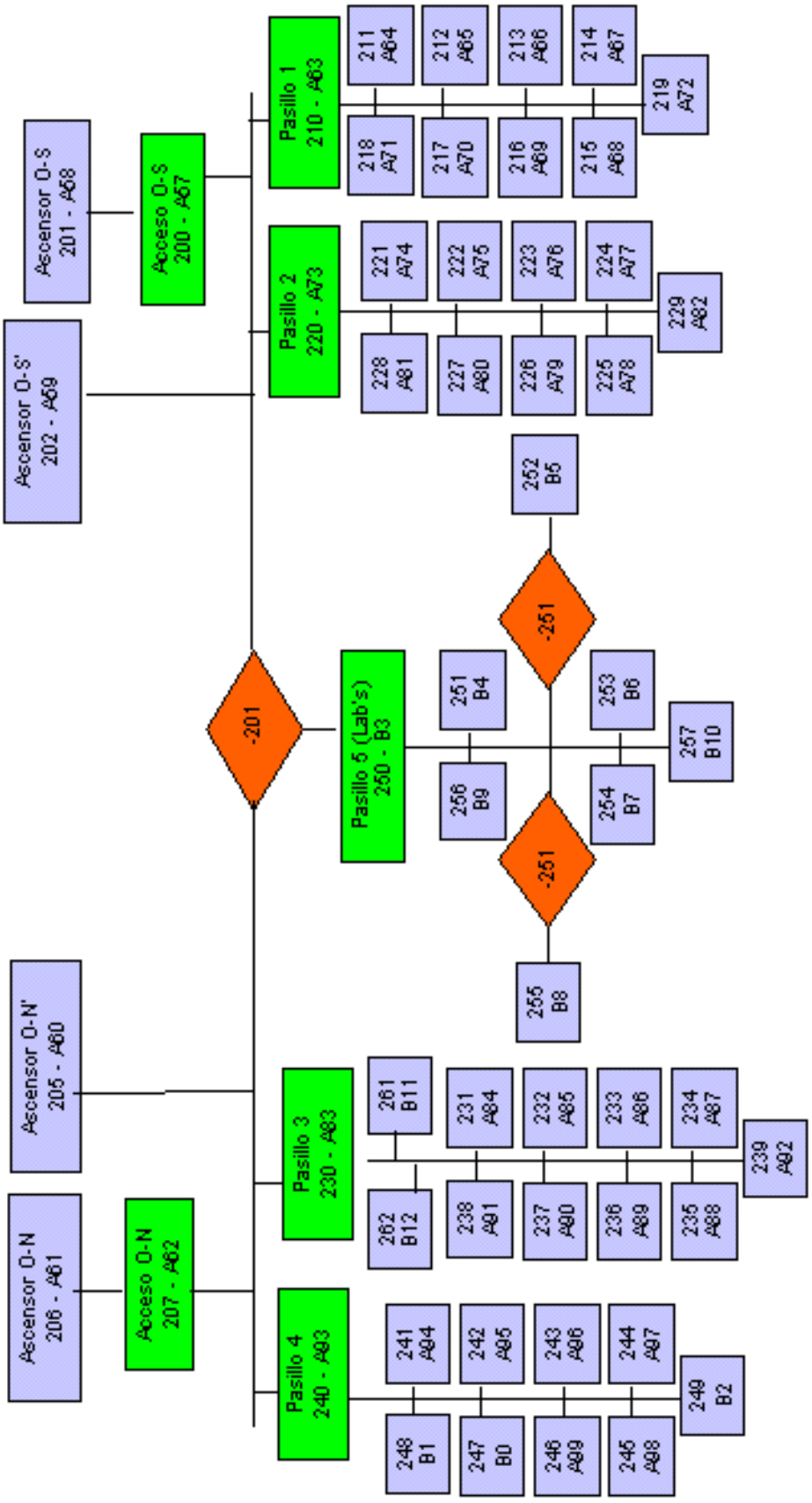


Figura 4.11. Arbol jerárquico de nodos en la planta segunda del edificio oeste de la Escuela Politécnica

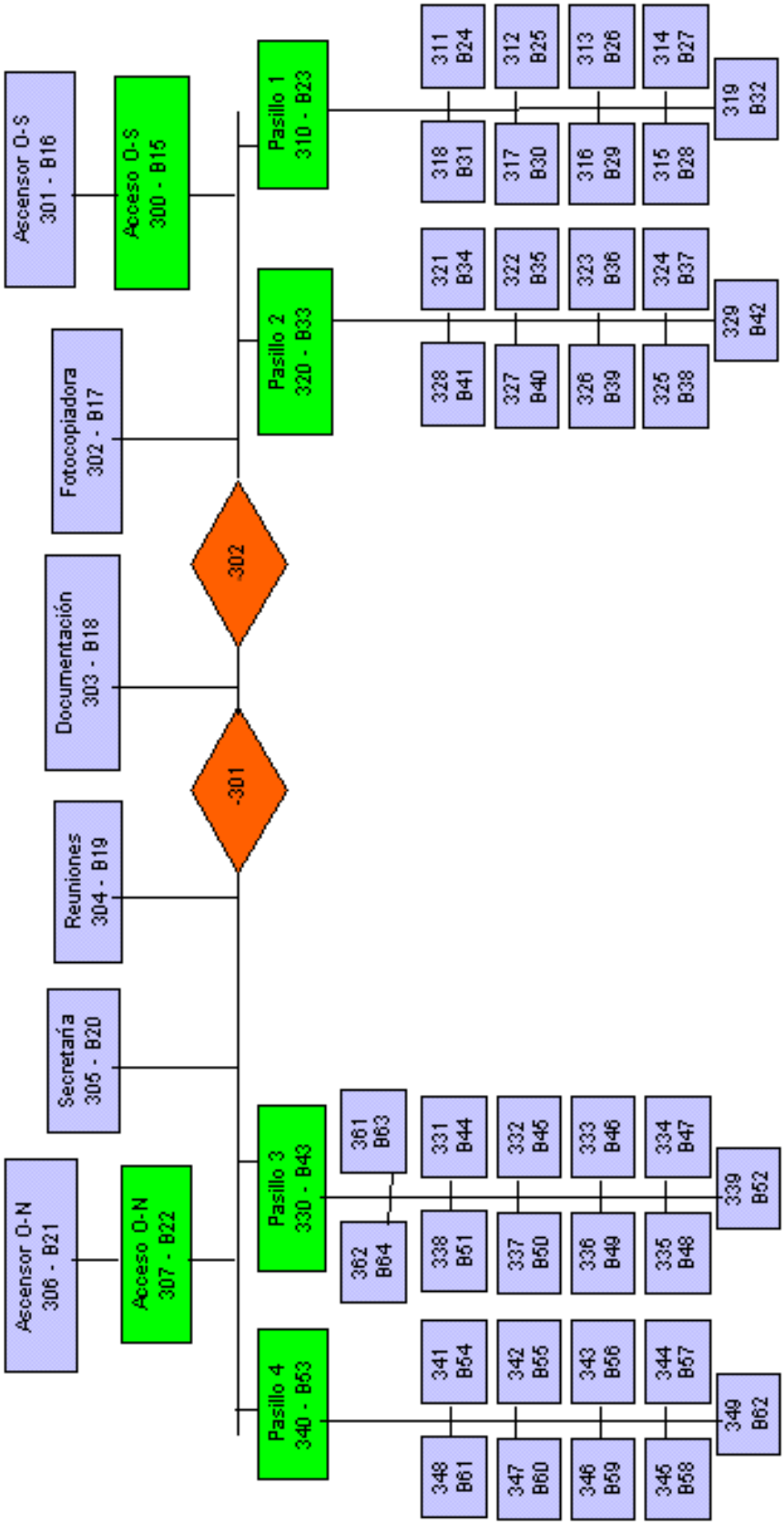


Figura 4.12. Arbol jerárquico de nodos en la planta tercera del edificio oeste de la Escuela Politécnica

A partir de los diagramas de las figuras 4.9, 4.10, 4.11 y 4.12 se construye el fichero ASCII que será el mapa empleado por los algoritmos de navegación como modelo de entorno. El mapa presenta solamente el campo de nombre de nodo, pues de momento ha sido el único descrito. A lo largo de la memoria se irán incluyendo nuevos campos y mostrando como va modificándose la apariencia del mismo.

Mapa parcial del ala oeste del Edificio Politécnico. Solo nombre de los nodos

	107	216	251	326
000	110	217	252	327
001	111	218	253	328
002	112	219	254	329
003	113	220	255	330
004	114	221	256	331
005	115	222	257	332
006	116	223	261	333
007	120	224	262	334
008	121	225	NE	335
010	122	226	NE	336
011	123	227	300	337
012	124	228	301	338
013	125	229	302	339
014	126	230	303	340
020	130	231	304	341
021	131	232	305	342
022	132	233	306	343
023	133	234	307	344
024	134	235	310	345
030	135	236	311	34
031	136	237	312	347
032	137	238	313	348
033	200	239	314	349
NE	201	240	315	361
NE	202	241	316	362
NE	205	242	317	-001
NE	206	243	318	-101
100	207	244	319	-131
101	210	245	320	-201
102	211	246	321	-251
103	212	247	322	-301
104	213	248	323	-302
105	214	249	324	
106	215	250	325	

Merece la pena recordar que el número de nodo no aparece en el mapa pues es el elemento que sirve para indexar el fichero. Es por ello que aparecen varios nodos cuyo nombre es NE. Con este código se indica al algoritmo de búsqueda que el nodo correspondiente a ese número no ha sido implementado, por lo que podría emplearse en futuras modificaciones en caso de ser necesario introducir un nodo en esa localización jerárquica.

Destacar simplemente, a la vista del mapa, su corta longitud, característica importante tal y como se comentó en apartados anteriores.

5. Gestión de Procesos.

La puesta en marcha de cualquier móvil autónomo requiere una serie de tareas que ya han sido comentadas anteriormente. Si bien existe multitud de formas de organizar el trabajo, el método tradicionalmente más usado en el campo de la robótica es el de la distribución jerárquica de tareas. Tal y como se observa en la figura 5.1 cada proceso ocupa una capa en la jerarquía, que proporciona resultados a las capas inferiores y pide servicios a las superiores, quedando el proceso que desarrollan transparente al resto de capas.

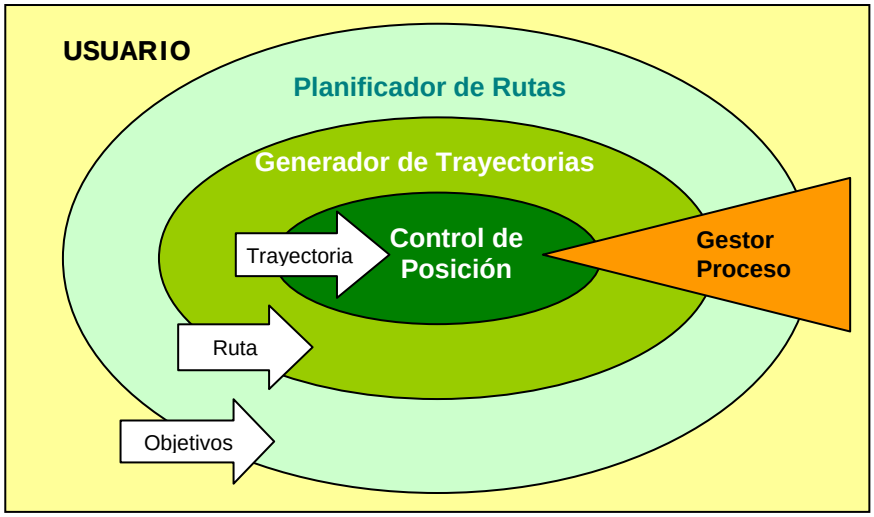


Figura 5.1. Organización en capas jerárquicas del proceso de navegación.

Se trata de una estructura multicapa típica en el diseño de aplicaciones software, pero incluye un proceso que solo aparece en aplicaciones ejecutadas en tiempo real, como los presentes en sistemas de control inteligente: un elemento encargado de supervisar el funcionamiento global del sistema, monitorizando variables, temporizando procesos y llevando la gestión de comunicaciones inter-proceso. Es la función del gestor de procesos.

El gestor de procesos es, por tanto, el organizador de la actividad global del sistema de navegación, el único proceso que tiene una visión global del movimiento de la silla de ruedas en cada instante.

5.1. Funcionalidad del gestor de procesos.

Tal y como se ha visto en la descripción general, las funciones fundamentales del gestor de procesos son dos:

1. *Controlar la entrada y salida del hilo de ejecución de cada uno de los procesos*. Debido a la funcionalidad de cada uno de ellos, cada proceso tiene un lugar distinto en el hilo de ejecución, y una secuencialidad distinta en el proceso global. El gestor de procesos se encarga de arrancar cada proceso en el momento adecuado, y con la periodicidad justa.

2. *Gestionar la comunicación entre los procesos.* Como se aprecia en la figura 5.1, la información intercambiada entre los procesos es distinta, pero también lo es el formato, y es necesario realizar un tratamiento previo. El gestor se encarga de esta tarea, bien sea de la realización física del mismo, como de la supervisión del proceso, según el caso.

Si bien ambas tareas están íntimamente relacionadas, en este capítulo se va a describir detalladamente cómo se realiza cada una de ellas por separado.

Tradicionalmente, en los sistemas de navegación autónoma, hay otras tareas que, debido a su visión global del sistema completo, también son realizadas por el gestor de procesos. Un buen ejemplo es la monitorización de tareas, procedimiento que le permite detectar si se ha quedado alguna tarea "colgada", al no responder durante cierto tiempo.

Todas estas particularidades de los sistemas multitarea ejecutados en tiempo real son perfectamente soportadas en casi todos los sistemas operativos diseñados para procesos que se ejecutan en tiempo real (tareas con un tiempo de respuesta crítico).

Se podría pensar, por tanto, que para esta aplicación sería interesante usar un sistema operativo de este tipo. Sin embargo realizando un estudio profundo de la organización del sistema se observa que no es imprescindible usar un núcleo operativo en tiempo real, por las siguientes razones:

1. No existen más que dos procesos cuya ejecución coincide en el tiempo: el gestor de procesos y el controlador de posición. La secuenciación de los mismos puede ser controlada perfectamente por el gestor de procesos.

2. Solo existe una tarea periódica realmente crítica, el controlador de posición, y su periodo de repetición no va a ser demasiado pequeño, teniendo en cuenta la dinámica del móvil (ver capítulo 2).

3. Al no haber procesos concurrentes, no hay problemas en la compartición de datos, y este aspecto es resuelto por el gestor a base de buffers lineales y circulares.

Por tanto, debido a la poca criticidad de las tareas, el uso de un núcleo en tiempo real prediseñado en lugar del gestor de procesos planteado no aporta una mejora significativa.

La figura 5.2 muestra el diagrama de flujo del funcionamiento del gestor de procesos. Se aprecia que existen dos partes distintas en el hilo de ejecución, una primera en la que se desarrolla el planificador de rutas, y una segunda que constituye un bucle infinito donde se desarrollan el generador de trayectorias y el controlador de posición.

En el diagrama se observa que el proceso completo finaliza al alcanzar el destino deseado. Si se desea realizar otro recorrido será necesario volver a arrancar el programa de navegación, indicando de nuevo origen y destino del movimiento a realizar.

Además de las tareas que a continuación se detallan en profundidad, el gestor de procesos se encarga también de la inicialización del sistema de comunicación con el bajo nivel. Mediante funciones del driver, el gestor realiza:

1. *La inicialización del puerto paralelo* a través del que se tiene acceso a la tarjeta de interfaz.
2. *El borrado de la memoria* en la que se implementa la comunicación. Se emplea para ello una memoria Dual Port.
3. *La liberación de los semáforos* que gestionan el acceso por ambas partes de la memoria.

Todos estos aspectos serán tratados en profundidad en el capítulo 9, donde se verán los detalles del sistema de comunicación con la red LonWorks de la silla de ruedas.

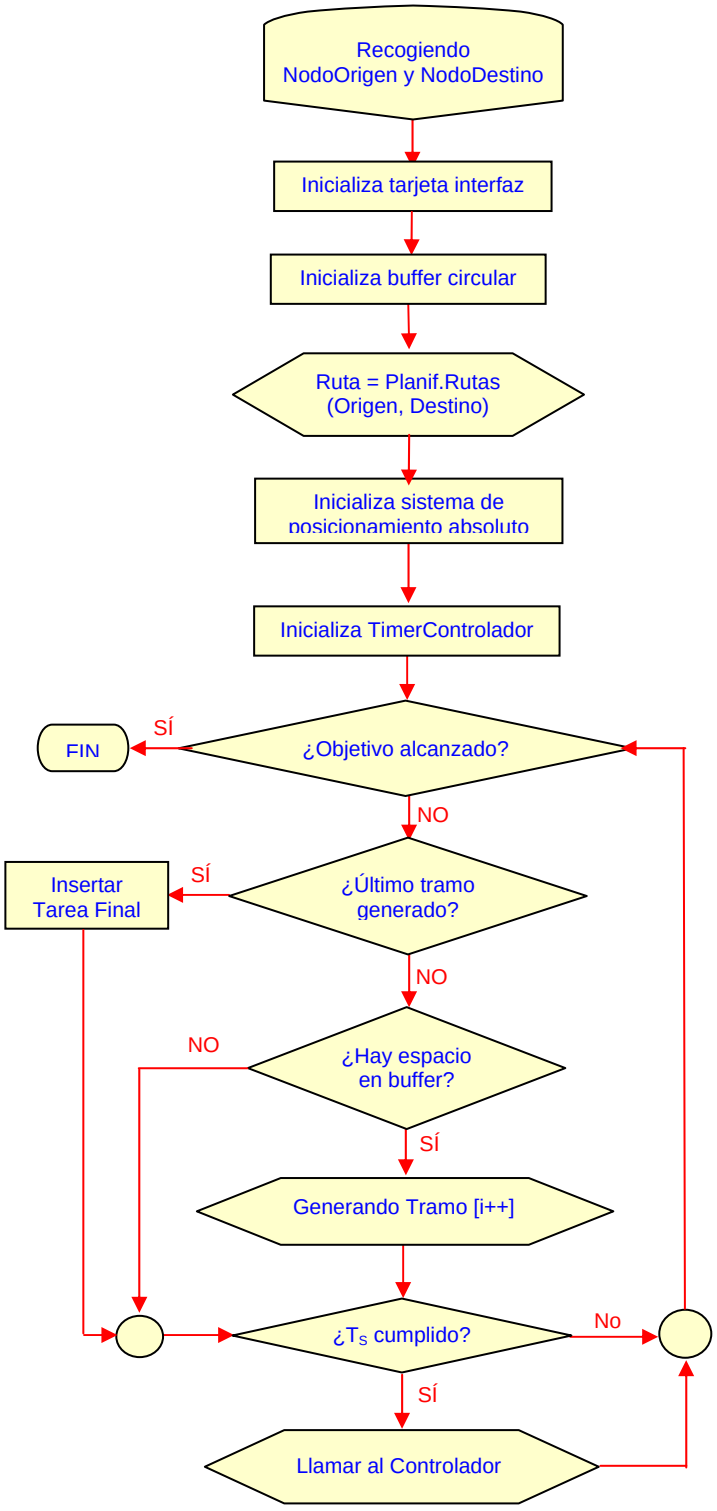


Figura 5.2. Diagrama de flujo de funcionamiento del gestor de procesos.

5.2. Organización de procesos.

Para poder analizar el trabajo del gestor en la organización de los procesos es necesario realizar primero una descripción de la localización de cada uno de ellos en el hilo de ejecución:

a) *El planificador de rutas.* Es el proceso encargado de proporcionar la ruta de nodos que une origen y destino, por lo que se ejecuta una sola vez y sin interrupción en el proceso de navegación, nada más comenzar el hilo de ejecución (ver figura 5.2). Se ha diseñado de este modo pues es un proceso corto y necesario para las tareas posteriores.

b) *El generador de trayectorias*. A partir del nombre de los nodos que forman la ruta, y con la información encontrada en el mapa de entorno, el generador obtiene la trayectoria más adecuada entre cada pareja de nodos mediante una secuencia de tareas, como se verá en el capítulo 7. Este proceso es algo más tedioso que el anterior, debido al cálculo que implica, por lo que, en vez de diseñarse de una sola vez todas las tareas que conforman la ruta completa, el proceso genera tantos tramos como sea posible en el hueco temporal que existe entre cada dos llamadas al controlador, tal y como se aprecia en la figura 5.2.

c) *El controlador de posición*. Se trata del único proceso periódico del sistema global, y como tal es llamado por el gestor cada período de muestreo. Como se ha comentado, su ejecución se intercala con la del generador de trayectorias (ver figura 5.2) que genera la consigna necesaria por el controlador para realizar su función.

Una tarea fundamental para la ejecución del controlador, y que realiza el gestor, pues es el único proceso que tiene una visión suficientemente amplia del proceso completo, es la inicialización del sistema de posicionamiento absoluto. Conociendo el nodo de origen y a través del mapa del entorno, el gestor obtiene las coordenadas x,y,θ de la silla.

5.3. La comunicación entre procesos.

Como se ha comentado, la otra tarea fundamental de la que se encarga el gestor es la de supervisar la comunicación entre los distintos procesos de navegación.

Tal y como se apreciaba en la figura 5.1, al margen del nodo de origen y de destino procedentes de la interfaz de usuario, existen dos elementos de comunicación fundamentales entre los procesos de navegación que serán comentados en los apartados siguientes: se trata de la ruta y la trayectoria.

5.3.1. La Ruta.

Procedente del planificador de caminos, la ruta es la secuencia de nodos por los que ha de pasar el móvil para alcanzar el destino. El método de almacenar la ruta es mediante un buffer lineal, cuyo puntero es controlado únicamente por el gestor.

Mediante este puntero, el gestor divide el proceso global en tramos (parejas de nodos pertenecientes a la ruta que constituyen origen parcial y destino parcial de la tarea diseñada por el generador de trayectorias). Son estos dos y no la lista completa de nodos lo que se proporciona al generador.

Este buffer ha sido dimensionado para dar cabida a 100 nodos, tamaño más que suficiente para la aplicación desarrollada. Además se trata de un buffer estático, pues al ser simplemente nombres de nodos lo que se almacenan en él, su tamaño no es suficientemente elevado como para tener que darle un tratamiento dinámico.

5.3.2. La Trayectoria.

Como se ha comentado en el punto anterior, los algoritmos de generación de trayectorias se incluyen en el bucle infinito de la aplicación junto con las tareas del controlador, de forma que mientras el móvil se encuentra trazando un tramo determinado, el generador obtiene la información necesaria para recorrer tramos posteriores.

Para sincronizar las tareas se emplea un buffer circular en el que el generador va depositando los parámetros que caracterizan a los tramos diseñados, y el controlador va extrayendo y eliminando la información según se emplea. El buffer circular se divide en celdas, conteniendo cada una de ellas la información relacionada con cada tramo.

Para recorrer con seguridad el buffer circular se emplean dos punteros, que son gestionados por cada proceso particularmente, pero son supervisados por el gestor.

a) *El puntero de tareas diseñadas:* Controlado por el generador de trayectorias, indica la celda sobre la que está trabajando o va a trabajar dicho proceso.

b) *El puntero de tareas descritas:* Controlado en este caso por el proceso de control de posición del sistema de navegación, indica el tramo que está recorriendo o acaba de recorrer la silla de ruedas.

En la figura 5.3 se muestra un diagrama explicativo del significado de los punteros.

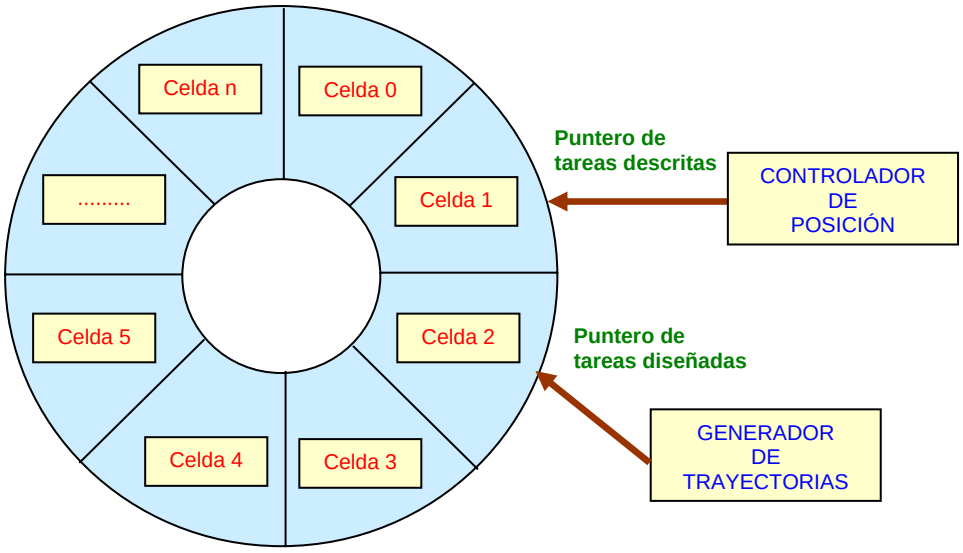


Figura 5.3. Diagrama explicativo de la gestión del buffer circular de comunicación entre el generador de trayectorias y el controlador de posición.

Cada una de las celdas del buffer circular es una estructura con el contenido necesario para generar la trayectoria asociada a la tarea en cuestión. En la figura 5.4 se muestra cuáles son los parámetros que componen la celda, así como una pequeña descripción de cada uno. En capítulos posteriores se observa la utilidad de cada uno de ellos, si bien aquí solo se pretende hacer una descripción funcional del buffer.

Parámetro	Identificación
x_i, y_i, θ_i	Coordenadas del punto inicial de la tarea
x_f, y_f, θ_f	Coordenadas del punto final de la tarea
x_c, y_c, R	Coordenadas del centro de curvatura en caso de tarea de giro, y valor del radio
STATUS o TIPO	Indica el tipo de tarea o el estado en que se encuentra la celda en el buffer.

Figura 5.4. Tabla de parámetros resultado de la generación de trayectorias.

El último campo de la estructura ayuda a la gestión del buffer, pues indica, entre otras cosas, cuándo la celda está vacía. Además ayuda también al controlador, indicando de qué tipo es la tarea que describe. En la tabla de la figura 5.5 se muestra el significado de los distintos valores de dicha variable:

Valor de STATUS	Significado
GIRO	Parámetros correspondientes a una tarea de giro.
AVANCE	Parámetros correspondientes a una tarea de avance.

FINAL	Es la siguiente tarea a la última de las que conforman la ruta completa.
VACIO	La celda está vacía.

Figura 5.5. Tabla explicativa del valor de la variable de estado.

Si bien el funcionamiento detallado del controlador se verá en el capítulo 8, es necesario realizar en este punto ciertas consideraciones sobre el mismo. Con el fin de hacer la conducción más cómoda para el usuario, la velocidad con la que avanza la silla responde a un perfil trapezoidal en cada tarea, de modo que al afrontar una nueva trayectoria el control adapta la velocidad de acuerdo con la tarea siguiente. Por tanto, para poder afrontar el diseño de una nueva trayectoria, el controlador necesita conocer los parámetros no sólo de la tarea en cuestión sino también de la siguiente.

Con la generación de trayectorias también es necesario tener en cuenta ciertas consideraciones. Ya se ha comentado que cada llamada al generador de trayectorias permite diseñar el camino que separa una pareja de nodos, lo que constituye un tramo. Si además se tiene en cuenta que, como ya se avanzó en el capítulo 3, la trayectoria que permite construir un tramo está formada por distintas tareas, se verá que para diseñar un tramo, el generador de trayectorias, empleará varias celdas en el buffer circular.

Sin entrar en mayor detalle (ver capítulo 7) se puede decir que cada tramo consta como máximo de tres tareas: una de salida del nodo de origen, otra de avance y una última de giro en el nodo de destino, es decir, hasta un máximo de tres tareas.

Todas estas consideraciones sobre los requerimientos de buffer de generador y controlador permiten dimensionar el tamaño del mismo, para lo cual se emplea la figura 5.6 modelo gráfico de lo explicado hasta el momento.

En la dicha figura se han incluido 3 celdas para incorporar los parámetros que caractericen la siguiente trayectoria a recorrer. Además se han añadido las dos celdas sobre las que estaría trabajando el controlador al mismo tiempo. Este sería el tamaño mínimo para el buffer circular (un total de 5 celdas), sin embargo si se desea que el generador pueda preparar la trayectoria del siguiente tramo sin esperar a que el controlador acabe con las tareas que acaba de diseñar es necesario incorporar otras 3 celdas, haciendo un total de 8.

Como el tamaño en bytes de cada celda (296 bytes teniendo en cuenta el tipo de los datos) no es demasiado, se elige como 10 el número de celdas del buffer circular, con el que se trabaja de forma cómoda.

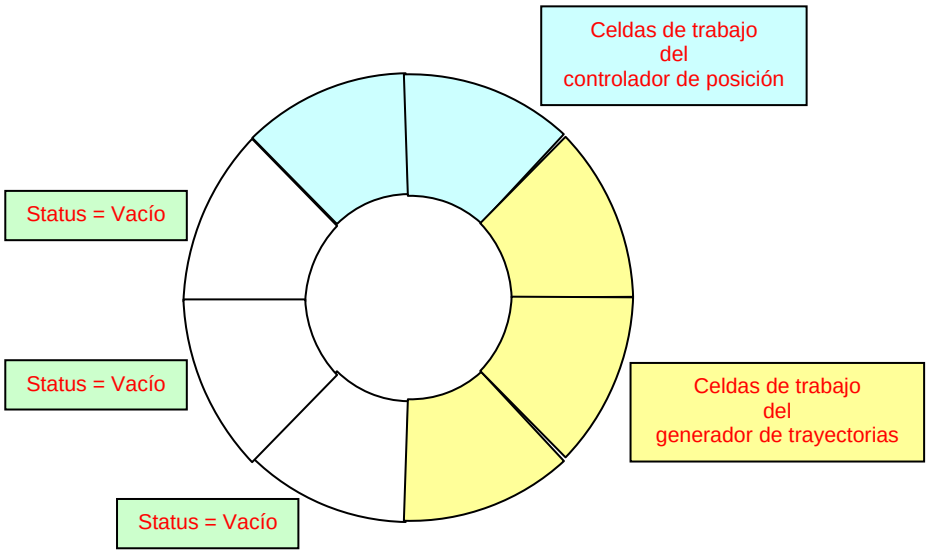


Figura 5.6. Diagrama aclaratorio de las necesidades de tamaño del buffer circular.

Visto todo lo anterior, se observa que el avance de los punteros en el buffer requiere cierto tratamiento especial.

a) *El avance del puntero de tareas descritas* lo realiza el propio controlador de posición. Este proceso indica con el puntero la tarea sobre la que está trabajando, y no realiza un avance del mismo hasta que no pasa a encargarse de la siguiente tarea, pues, como ya se comentaba, el controlador de posición necesita para realizar el control de la tarea n, los parámetros de la n+1.

   1. El controlador comprueba que, tanto la tarea siguiente como la posterior a la siguiente ya han sido generadas, consultando la variable de estado de dichas celdas.
   2. Hecho esto, el controlador libera la celda de tarea que ocupaba, para lo cual invalida la variable de estado de la celda, tal y como se veía anteriormente.
   3. Finalmente el puntero pasa a la posición de la tarea siguiente.

b) *En cuanto al puntero de tareas generadas*, también es el propio proceso de generación de trayectorias el que controla su avance, pero en este caso la gestión de cuándo avanzar la realiza el gestor, que no arrancará el proceso de generación hasta que no existan al menos tres celdas libres en el buffer circular.

Finalmente, queda por comentar que es el gestor el encargado de controlar el final del proceso completo de navegación, para lo cual se hace uso de una tarea final y de un flag de finalización:

1. Tras generarse la trayectoria que lleva al nodo de destino seleccionado por el usuario, el gestor añade una tarea final cuyo único campo útil es el de estado, donde se indica al controlador que se trata de la última tarea.

2. El controlador a su vez activa un flag cuando ha cubierto la última trayectoria y ha parado la silla, con lo que el gestor finaliza la ejecución del navegador completo.

Todos los aspectos aquí resaltados serán, en cualquier caso, revisados de nuevo en el capítulo relacionado directamente con el procesador concreto (ver capítulos 7 y 8).

6. Planificación de Rutas.

Los algoritmos de planificación de rutas permiten obtener el camino óptimo para llevar a un sistema desde un punto de origen a otro de destino. Para conseguirlo el algoritmo se basa en un modelado del entorno concreto, que determina en gran medida el algoritmo a emplear y el tipo de salida que proporciona.

Gracias a la organización nodal desarrollada para el entorno de movimiento bajo estudio (ver capítulo 4, Modelado del Entorno) el algoritmo que genera la ruta del movimiento es muy sencillo en este caso, pues se sirve de la información de jerarquía implícita en el mapa para obtener como salida una lista de nodos que permita unir el origen y el destino solicitados por el usuario.

6.1. Algoritmo basado en árbol jerárquico de nodos.

Como ya se ha comentado, el algoritmo de planificación está directamente relacionado con el modelado del entorno elegido. En el capítulo anterior se mostró que en este proyecto se ha empleado un sistema nodal de modelado de entorno; para estos casos se suelen emplear distintos algoritmos de búsqueda de la secuencia de nodos que proporciona el camino óptimo entre origen y destino en base a distintas características (número de nodos, distancia euclídea total, curvatura de la trayectoria a seguir, etc.).

El más conocido es el algoritmo de Dijkstra, si bien cualquier algoritmo de inteligencia artificial de búsqueda del camino más corto en un árbol no necesariamente binario sería igualmente adecuado.

Sin embargo en este trabajo se cuenta con la información adicional de la posición jerárquica de los nodos. El árbol a recorrer, por tanto es un árbol fuertemente jerarquizado, por lo que los caminos posibles entre nodo y destino no van a ser muchos. Esta característica es una constante en los entornos estructurados para los que se ha diseñado el sistema.

En la figura 6.1 se muestra el grafo correspondiente al diagrama presentado en la figura 4.8 correspondiente al modelo de la planta baja del edificio. En esta figura se observa la jerarquía nodal y el alto grado de estructuración comentado.

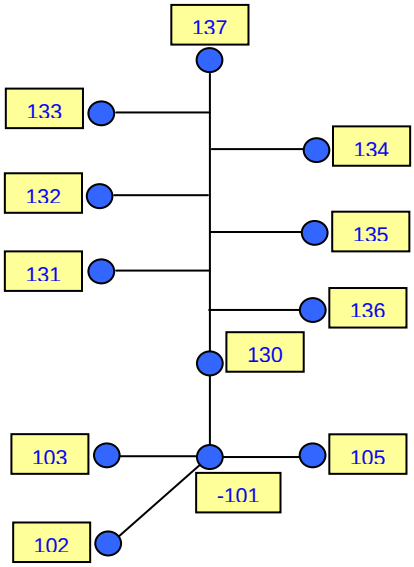


Figura 6.1. Diagrama de nodos de la planta baja del ala oeste del Edificio Politécnico.

El algoritmo simplemente compara los nombres jerárquicos de los nodos de origen y destino, de modo que en caso de que no exista coincidencia en alguno de los tres niveles jerárquicos se añade a la ruta los nodos que permiten cambiar de jerarquía. Además comprueba cuándo es necesario añadir un nodo de transición, en función del nivel jerárquico en el que se mueva. El funcionamiento del algoritmo se muestra en el diagrama de bloques de la figura 6.2:

Recordando los tipos de nodos que se pueden encontrar en el mapa del entorno, la lista de nodos que forman la salida del planificador tiene dos únicos nodos de tipo sala, que serán el nodo origen y el nodo destino, ya que el resto serán nodos jerárquicos y de transición. Es por ello que la lista de nodos nunca va a ser muy larga, cosa que por otro lado tampoco es necesaria, teniendo en cuenta que se desea un movimiento del robot más cómodo que exacto. Un número alto de subobjetivos intermedios definiría más exactamente la posición del móvil en cada instante, pero provocaría un mayor tiempo de reacción del mismo con lo que el movimiento pierde agilidad y comodidad.

Tal y como se observa en la figura 6.2, solamente en caso de aparecer un cambio de jerarquía en el nivel más alto, será necesario elegir entre los distintos caminos que

pueden aparecer. En este caso se ha decidido emplear un algoritmo de búsqueda en profundidad (el criterio de selección es la distancia euclídea total del recorrido) pues es el más sencillo y no existen en esta aplicación otros criterios con más peso.

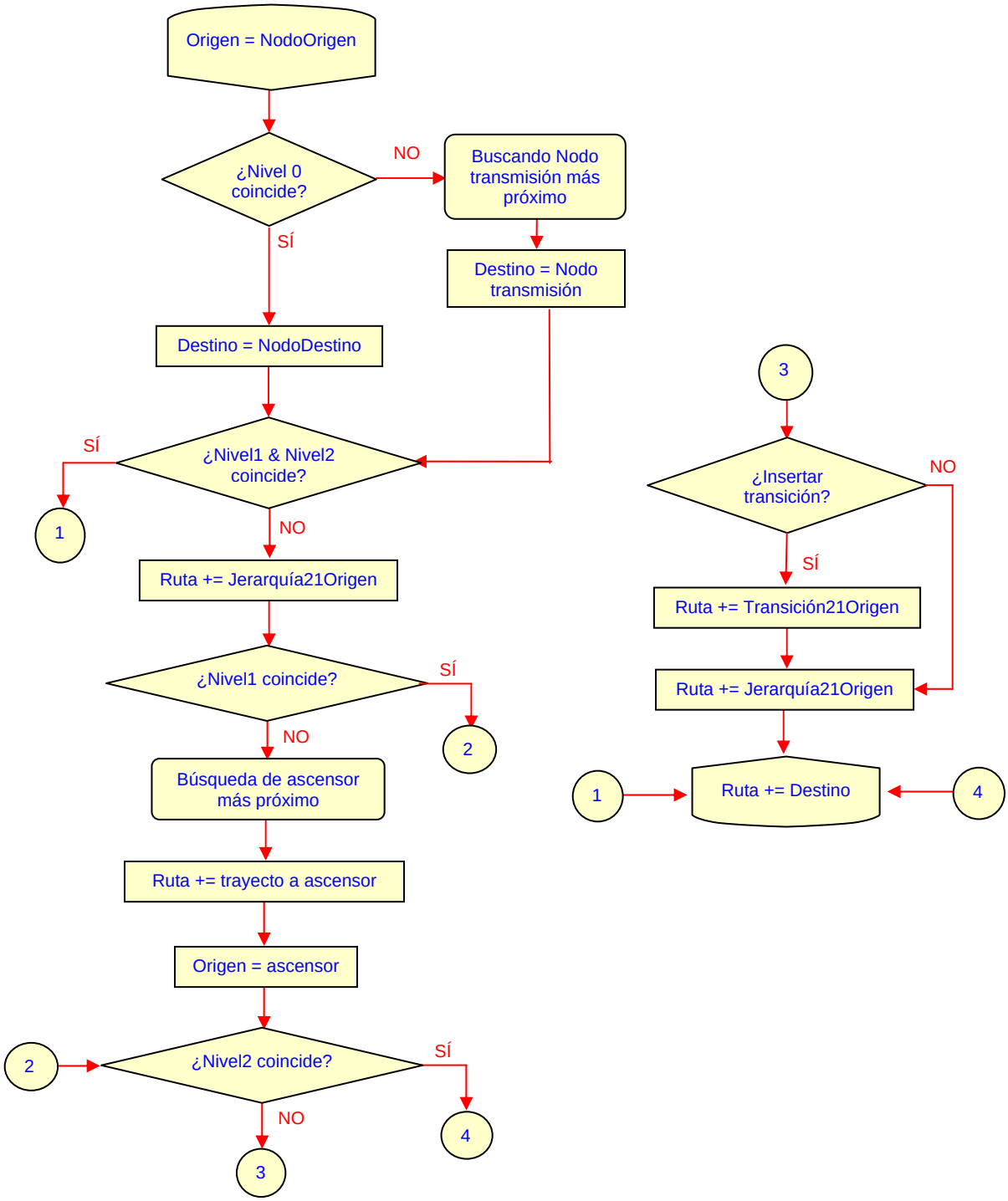


Figura 6.2. Algoritmo de planificación de rutas basado en árbol jerárquico de nodos

El algoritmo requiere como entrada el nombre del nodo origen y destino proporcionados por la interfaz de usuario. En este caso, el nombre incluye la cifra que identifica al mapa dentro del Edificio Politécnico, de modo que si es necesario, el generador diseñará el trayecto de cambio de mapa, como se comentará más adelante.

Como salida, este módulo proporciona una lista con los nombres de los nodos de la ruta que ha de seguir la silla para alcanzar el destino final.

Como se ha comentado, en muchos casos no va a ser necesario realizar una selección del camino más corto, sino que en mapas tan estructurados como el que se trata en este proyecto, existe un único camino para ir de un lugar a otro, o, si existen varios, suelen ser de la misma longitud. Debido a ello, en la mayor parte de los casos, el planificador no requiere del mapa para generar el camino, sino que le basta con el nombre de los nodos para recorrer de forma óptima el árbol jerárquico. Esta es una de las características más atractivas del sistema propuesto, y es que el esfuerzo off-line realizado a la hora de modelar el mapa de forma jerárquica se ve recompensado ya en la tarea de planificación de rutas.

6.2. Algoritmo de cambio de planta.

Como ya se ha comentado, los propios nodos proporcionan sin ayuda de ningún mapa su localización jerárquica en el entorno. Es por ello que el mapa sólo se emplea cuando existen varios posibles caminos para unir el origen y el destino, y existe gran diferencia en el factor de mérito elegido, en este caso la distancia total a recorrer.

El único caso en que es necesario realizar este tratamiento en el edificio bajo estudio es cuando la trayectoria incluye un cambio en el nivel jerárquico más alto (un cambio de planta a través de un ascensor): se trata de buscar el ascensor óptimo.

Para realizar la selección, es necesario conocer la posición de los nodos que forman los posibles caminos para obtener y comparar un factor de mérito asociado a cada uno de ellos, que informa de la distancia euclídea global de todos los trayectos. El factor elegido es el que se muestra en la ecuación 6.1, en la que i representa al nodo i-esimo de cada una de las rutas posibles.

$$FM = \sum_{i=0}^{i=n} \left[(x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2 \right] \qquad <6.1>$$

Tal y como se observa, el factor de mérito no coincide con la distancia global a recorrer por el móvil, pero sirve para discriminar el camino más corto. Se podrían haber elegido otros factores que incorporasen información sobre la dificultad de uno u otro trazado (pasillos más estrechos, mayor número de curvas) pero tampoco tiene sentido, pues la dificultad presentada por todos los posibles trayectos es semejante en el caso concreto de la aplicación real bajo estudio.

En la figura 6.3 se muestra un ejemplo del camino elegido por el planificador de caminos en distintas a partir de distintos orígenes, pudiendo comprobar el funcionamiento del algoritmo del ascensor óptimo.

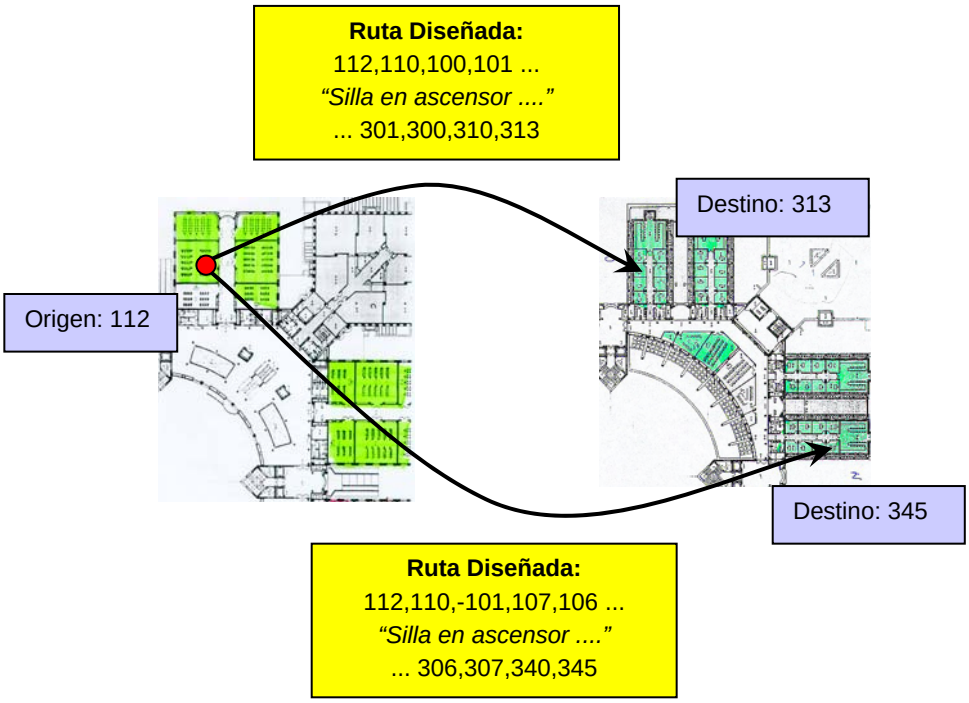


Figura 6.3. Ejemplo de selección de ascensor óptimo.

6.3. Algoritmo de cambio de ala.

Como ya ha sido comentado anteriormente (capítulo 4), en el modelado del entorno de trabajo del proyecto cada ala del Politécnico se corresponde con un mapa distinto, facilitando de este modo la codificación jerárquica de nodos. Sin embargo esto no constituye restricción alguna para que el usuario pueda moverse por todo el edificio, pues se incorpora la capacidad de que el usuario pueda elegir como destino un nodo que no pertenezca al mapa en que se encuentra por encontrarse en otra ala.

De este modo se aprovecha la característica de que el mapa se encuentra autocontenido en el edificio, por lo que cambiar de entorno no conllevará mayor problema que el de capturar el nuevo mapa. Este método se puede emplear en cualquier otro entorno estructurado en el que se aplique el sistema, sin más que realizar un correcto modelado del entorno, y con ello se consigue que el mapa sea más pequeño con la consecuente mayor agilidad de los algoritmos de navegación.

Como ya se comentó, para indicar al planificador que el usuario desea acceder a otro mapa, la interfaz de usuario incluye en el nombre de los nodos origen y destino un campo que informa del mapa al que se refieren los nodos. El planificador es el único

elemento de todo el sistema de navegación que va a contar con esta información (transparente para el resto del sistema), que se trata de la siguiente forma (ver figura 6.2):

- Compara al campo de mapa del nombre del nodo origen con el de destino para ver si coinciden.
- En caso de coincidir, obtiene la lista de nodos que conforman el camino más corto entre origen y destino.
- En caso de no coincidir, ignora el nombre del nodo destino, eligiendo como tal un nodo intermedio en el que se obtendrá la información necesaria para retrazar la ruta hasta el destino. Para conseguirlo el sistema realiza los siguientes pasos:

    1. Realiza la selección del sentido en el que se van a recorrer las alas del edificio, intentando siempre minimizar la longitud del camino.
    2. Seguidamente, el móvil se dirige a la salida del mapa actual que ha elegido el planificador, donde el sistema recoge el mapa del nuevo entorno.
    3. En caso de que el mapa destino sea el nuevo al que ha accedido el móvil, se retraza entonces la trayectoria hasta el nodo de destino elegido.
    4. En caso contrario, el proceso se vuelve a repetir en el nuevo mapa al que se ha accedido hasta que el sistema se encuentre en el ala de destino y posea el mapa con el que obtener la ruta final.

El algoritmo de selección del punto de salida del mapa de origen es semejante al ya realizado para el cambio de planta, y para implementarlo se utiliza, de nuevo, el factor de mérito de la distancia mínima.

En la figura 6.4 se muestra un ejemplo del funcionamiento del algoritmo de cambio de ala.

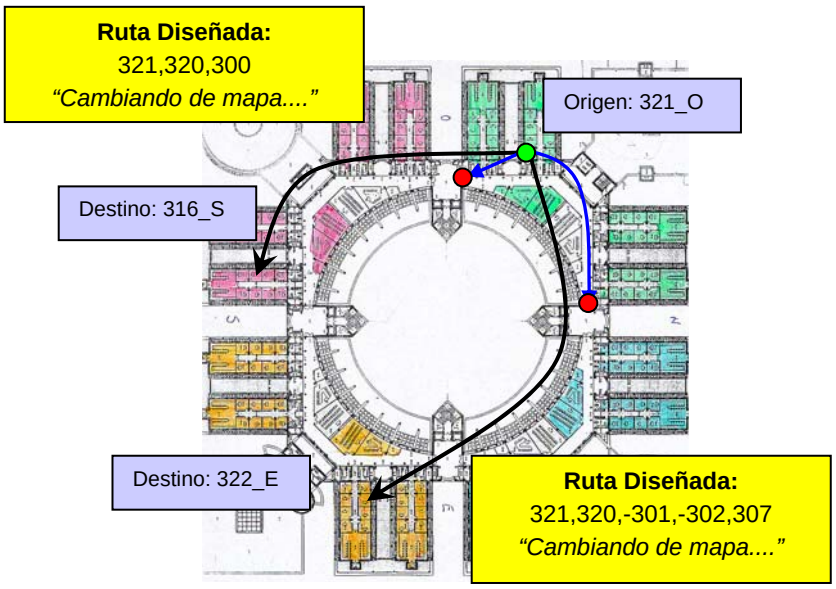


Figura 6.4. Ejemplo de funcionamiento del algoritmo de cambio de ala.

En este aspecto el software desarrollado recuerda un poco al sistema de encaminamiento de los routers en una red de ordenadores como puede ser Internet. El router no proporciona el camino hasta al destino porque no lo conoce, sino que simplemente mediante un sistema de jerarquización (distintos dominios, subredes, etc.) genera la ruta hasta el destino dentro del nivel jerárquico que encabeza o, si no se encuentra en el mismo nivel que el destino, envía el paquete a otro router que conozca el camino.

6.4. Nodos transmisores del mapa.

Una de las bases sobre la que se fundamenta la aportación del proyecto es el hecho de que el sistema de navegación se base en un mapa contenido en el propio edificio. Como mapa autocontenido, la portabilidad del sistema de navegación es máxima, ya que cuando el móvil cambia el entorno en el que encuentra, recoge del nuevo entorno la información necesaria para desenvolverse en ese nuevo medio.

Será entonces necesario contar con lo que se ha llamado "nodos transmisores de mapa". Se trata de colocar elementos activos próximos a nodos concretos del mapa por los que se pasa al acceder al nuevo ala. Estos elementos, mediante un protocolo que se debe preestablecer proporcionan al móvil la información necesaria para desenvolverse en el entorno al que accede.

La elección de los nodos que se comporten como transmisores de mapa depende del entorno de trabajo concreto, pues parece lógico pensar que se deberá colocar un nodo transmisor de mapa en cada acceso al nuevo edificio, y, observando el plano de cualquier edificio se aprecia que suele haber más de un acceso distinto.

En el caso del Edificio Politécnico, existen múltiples accesos a cada una de las alas que constituye un mapa distinto. Concretamente, los nodos por los que un posible usuario puede acceder a cada mapa cumplen una serie de características claras, basadas en la especificidad del problema planteado:

- Solo son válidos aquellas entradas que incluyan un mínimo de accesibilidad pra un móvil rodado.
- El acceso puede producirse desde el exterior del edificio o desde otro mapa en el interior del propio edificio.
- Para el caso de que el acceso se realice desde fuera del edificio se han de colocar nodos transmisores en el las puertas de acceso desde el exterior.
- En el caso de cambio de mapa desde el interior del edificio, los accesos pueden estar en cualquiera de las plantas, pues hay comunicación entre alas en todas las plantas.

Solamente los nodos mostrados en el siguiente mapa (figura 6.5) cumplen esas características y, por tanto, será aquí donde se tengan que instalar los nodos transmisores.

Figura 6.5. Localización de nodos transmisores de mapa del edificio oeste de la Escuela Politécnica

Como nodo transmisor se ha pensado en el uso de un transceiver de radio, que es fácilmente implementable y dan menos problemas que, por ejemplo, la transmisión infrarroja, cuando hay objetos que obstaculizan el camino visual entre el emisor y la silla. Este punto no es de interés en el proyecto, pero sería necesario implementarlo para poner en marcha el conjunto.

6.5. Origen de coordenadas del mapa de entorno.

Tal y como se ha observado en puertos anteriores, existen casos en los que es necesario conocer la localización exacta de los nodos del mapa, en principio, para que el planificador de caminos obtenga la ruta al destino más adecuada.

Aparece, por tanto, la necesidad de fijar cuál va a ser el origen de coordenadas del entorno bajo estudio, para lo cual se van a plantear las siguientes condiciones:

a) Este punto ha de depender del mapa que se trate, es decir, del ala donde se encuentre el móvil en cada momento en el caso concreto de la aplicación sobre el Edificio Politécnico.

b) Generalmente y por simplificar los algoritmos, los sistemas de coordenadas elegidos en los trabajos de navegación por interiores suelen ser de dos dimensiones, con lo que la posición del nodo también va a ser relativa a la planta del edificio donde se encuentre, ya que no se codifica la dimensión z.

En este caso particular se sigue la norma y se utiliza un sistema de coordenadas bidimensional, teniendo en cuenta que un sistema de posicionamiento de tres dimensiones sería redundante, pues la información sobre la planta en que se encuentran los nodos ya aparece en el nombre del nodo, y una mayor exactitud en su medida de la tercera dimensión carece de interés en los algoritmos de trayectometría.

c) Debido a la simetría que suelen presentar todos los edificios algo estructurados en su organización en plantas, es posible elegir como origen de coordenadas un punto localizado en todas las plantas con el mismo par (x,y), con lo que el algoritmo del planificador de caminos se simplifica notablemente.

d) También simplifica el tratamiento, el hecho de que el origen de coordenadas sea un nodo del propio mapa, y por lógica uno de los nodos transmisores de mapa, ya que es en ellos en los que se transfiere el mismo. Como generalmente existen varios puntos de ese tipo es necesario elegir uno como origen de coordenadas, y en cada caso particular la elección del mismo vendrá justificada por la organización jerárquica del edificio.

Todos los planteamientos anteriores llevan a la conclusión de que el origen de coordenadas será uno de los nodos transmisores de mapa presentados en la figura 6.5, y que ha de ocupar la misma posición (x,y) en todas las plantas.

Un planteamiento lógico llevaría a pensar que, las entradas al edificio desde el exterior serían los puntos más idóneos para ser origen de coordenadas. Sin embargo, en el caso práctico objeto de estudio, las entradas al edificio no tienen salas en puntos (x,y) equivalentes en otras plantas distintas de la planta baja y no presentan simetría ninguna en cuanto al ala se refiere (se sitúan en distintos sitios en cada ala), por lo que no resultan candidatos adecuados.

Sin embargo, si se observa detalladamente el mapa se aprecia que, una vez dentro del edificio, el cambio de ala generalmente se hace a través de los distribuidores entre plantas, a los que desembocan los ascensores. Se elegirán, por tanto, los nodos localizados en estos puntos como origen de coordenadas del mapa.

Empleando los argumentos anteriores se diseña, para la aplicación concreta bajo estudio, el siguiente mapa (figura 6.6), en el que se muestran cuáles han sido los puntos origen de coordenadas en cada una de los mapas del Edificio Politécnico. Aunque solo se muestra una planta, tal y como se ha comentado el origen tiene la misma posición (x,y) absoluta en todas ellas.

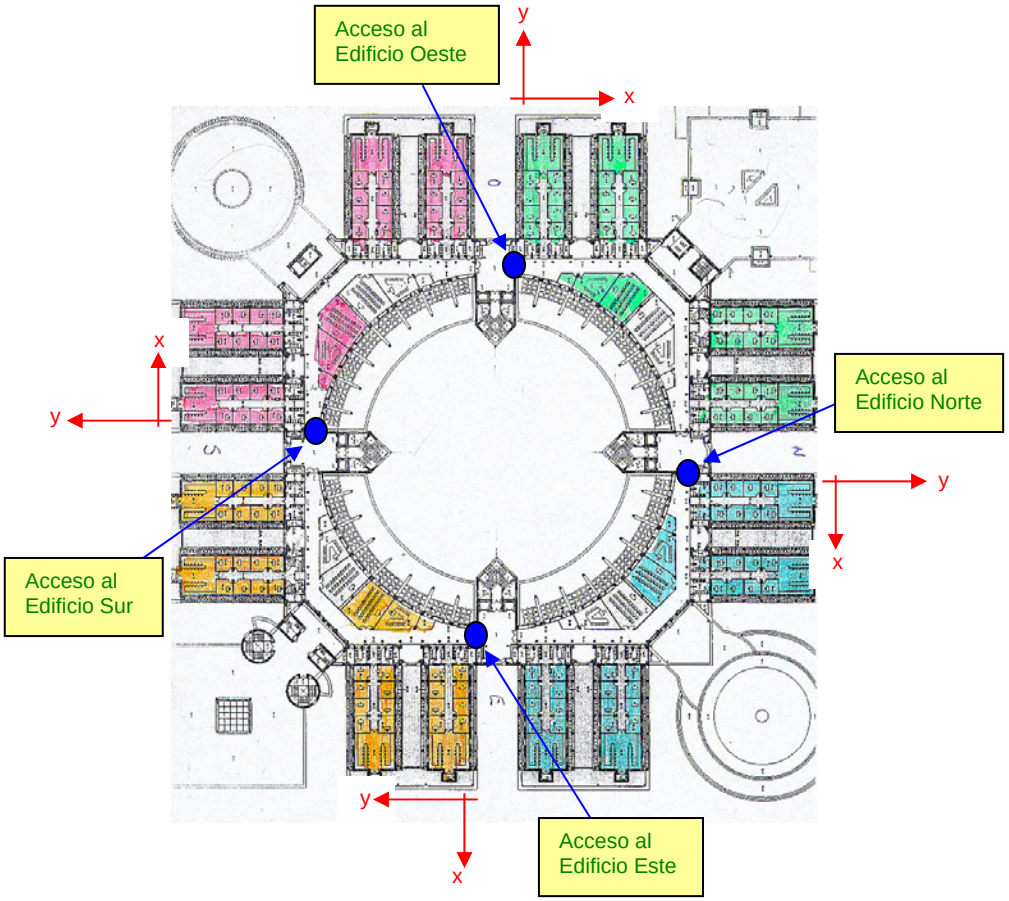


Figura 6.6. Organización del sistema de coordenadas de posición absoluta para cada edificio (planta baja).

El hecho de que se haya elegido, dentro de ser un distribuidor entre plantas y alas, ese y no otro el nodo que se comporte como origen de coordenadas es algo aleatorio, que no influye en el funcionamiento del algoritmo final.

6.6. Mapa del entorno para el planificador de caminos.

Una vez estudiado el funcionamiento completo del planificador de rutas, es posible completar parcialmente el contenido del mapa con el que va a trabajar el sistema de navegación.

En el mapa, al nombre de los nodos, presentado en el capítulo 4 se le añade la posición de cada nodo, relativa al origen de coordenadas fijado en la figura 6.6 para el edificio oeste.

La nueva leyenda del mapa correspondiente al ala oeste del Edificio Politécnico que se muestra a continuación sería:

Nombre_del_nodo coordenada_relativa_x coordenada_relativa_y.

Mapa parcial del ala oeste del Edificio Politécnico. Nombre y posición relativa de los nodos

```
000 0 0
001 -0.3 -1
002 3.7 -0.7
003 0.5 0.5
004 7 -6
005 5.7 -2.7
006 5.6 -6.7
007 6.5 -6.5
008 2.8 -3
010 2.5 0.5
011 2 3
012 2 5
013 4.3 3
014 3.3 3
020 7 -3.7
021 ?
022 ?
023 ?
024 ?
030 6 -0.5
031 9 2.7
032 9.5 2.4
033 10.5 4
NE
NE
NE
NE
100 0 0
101 -0.3 -1
102 3.7 -0.7
103 0.5 0.5
104 7 -6
105 5.7 -2.7
106 5.6 -6.7
107 6.5 -6.5
110 2.5 0.5
111 2.2 2.2
112 2.2 3.6
113 2.2 5
114 3 5
115 3 3.6
116 3 2.2
120 7 -3.6
121 9 -3.4
122 10.4 -3.4
123 11.8 -3.4
124 11.8 -4.2
125 10.4 -4.2
126 9 -4.2
130 6 -0.5
131 8.2 3
132 9 4
133 10.3 4.3
134 10.8 3.8
135 10.7 2.9
136 8.5 2.7
137 10.8 4.3
200 0 0
201 -0.3 -1
202 3.7 -0.7
205 5.7 -2.7
206 5.6 -6.7
207 6.5 -6.5
210 1.1 0.5
211 1 1.8
212 1 2.2
213 1 2.9
214 1 3.6
215 1.3 1.5
216 1.3 2
217 1.3 2.7
218 1.3 3.3
219 1.1 3.9
220 4.2 0.5
221 4 1.5
222 4 2
223 4 2.7
224 4 3.3
225 4.3 1.8
226 4.3 2.2
227 4.3 2.9
228 4.3 3.6
229 4.2 3.9
230 7.1 -2.3
231 8.9 -2.2
232 9.3 -2.2
233 10.1 -2.2
234 10.8 -2.2
235 8.6 -2.5
236 9.1 -2.5
237 9.8 -2.5
238 10.4 -2.5
239 11.1 -2.3
240 7.1 -5.3
241 8.6 -5.2
242 9.1 -5.2
243 9.6 -5.2
244 10.4 -5.2
245 8.9 -5.5
246 9.3 -5.5
247 10.1 -5.5
248 10.8 -5.5
249 11.1 -5.3
250 6.1 -0.4
251 8.2 1.9
252 9 4
253 10.3 4.2
254 10.9 3.6
255 10.7 2.3
256 8.5 1.5
257 10.9 4.2
261 8.1 -2
262 8.1 -2.7
NE
NE
300 0 0
301 -0.3 -1
302 2.2 -0.2
303 5.5 -0.7
304 6.4 -2
305 6.4 -3.9
306 5.6 -6.7
307 6.5 -6.5
310 1.1 0.5
311 1 1.8
312 1 2.2
313 1 2.9
314 1 3.6
315 1.3 1.5
316 1.3 2
317 1.3 2.7
318 1.3 3.3
319 1.1 3.9
320 4.2 0.5
321 4 1.5
322 4 2
323 4 2.7
324 4 3.3
325 4.3 1.8
326 4.3 2.2
327 4.3 2.9
328 4.3 3.6
329 4.2 3.9
330 7.1 -2.3
331 8.9 -2.2
332 9.3 -2.2
333 10.1 -2.2
334 10.8 -2.2
335 8.6 -2.5
336 9.1 -2.5
337 9.8 -2.5
338 10.4 -2.5
339 11.1 -2.3
340 7.1 -5.3
341 8.6 -5.2
342 9.1 -5.2
343 9.6 -5.2
344 10.4 -5.2
345 8.9 -5.5
346 9.3 -5.5
347 10.1 -5.5
348 10.8 -5.5
349 11.1 -5.3
361 8.1 -2
362 8.1 -2.7
```

Las coordenadas aparecen escaladas en cm con una escala de 1:500.

7. Generación de Trayectorias.

Los algoritmos de generación de trayectorias permiten obtener de forma funcional o paramétrica la descripción de la trayectoria necesaria para unir dos o más puntos de un camino.

Tradicionalmente este módulo era realizado mediante algoritmos más o menos complejos que permiten unir mediante una curva dos posiciones de un plano bidimensional (x,y). Se trata de los algoritmos de bsplines, clotoides, y sobre todo de splines de grado n, que es el empleado más reiteradamente.

En resumidas cuentas, el objetivo perseguido es siempre el mismo: obtener la ecuación de la trayectoria que ha de seguir el móvil a partir de unas premisas de puntos inicial y final, curvatura, etc. Esta ecuación será empleada por el controlador de trayectorias para obtener la consigna de posición del móvil en cada periodo de muestreo.

7.1. Elección del algoritmo de generación de trayectorias.

La elección de la forma de la trayectoria, y, por tanto, del algoritmo de generación de trayectorias, se basa generalmente en conseguir la mayor suavidad posible en el movimiento. Se observa, por ejemplo que en el caso de la spline de orden mayor a 3 la curva se vuelve muy sinuosa, con lo que no se emplea.

Este tipo de algoritmos resultan especialmente útiles a la hora de incorporar on-line cambios en la trayectoria prediseñada por la aparición de obstáculos en el camino. Como este aspecto no va a ser tenido en cuenta en el proyecto, la generación de trayectorias mediante splines pierde parte de su interés.

Además tampoco las curvas splines de tercer grado son a veces la trayectoria más adecuada para el movimiento de una silla en interiores estructurados: ¿tiene sentido generar una curva para recorrer el pasillo de un hospital?

Como solución alternativa en este proyecto se plantea realizar el movimiento del robot en base a líneas rectas y tramos de curva. Algunas de las razones para emplear este y no otro modelo de trayectoria son las siguientes:

- En el afán de hacer el movimiento de una silla de ruedas lo más humano posible, los comandos de *"avanzar en línea recta desde el punto x_i y_i al punto x_f y_f* " o *"girar hasta estar en la dirección θ con radio R"* son movimientos más naturales que los generados por una spline, por lo que el usuario se sentirá más cómodo con la conducción.

- Al realizar las trayectorias mediante rectas y segmentos de circunferencia, el avance entre dos puntos se divide en segmentos más cortos, con lo que el problema global de la tarea del generador se subdivide, y de este modo se simplifica frente a la carga computacional de las otras.

- Las curvas clotoidales y splines no respetan la dinámica del móvil a no ser que se establezcan ciertas restricciones, con lo que de nuevo se complica el algoritmo, mientras que con el modelo planteado es muy sencillo imponer restricciones de diseño al generar la trayectoria.

- La mayor parte de los giros en entornos interiores como los comentados son de 90 grados y en muchos casos se requieren giros de 180 grados para cambiar de sentido, en el avance a lo largo de un pasillo. Estas trayectorias que implican giros sobre sí misma del robot no son implementables mediante una spline, mientras que sí lo son con este nuevo método.

En la figura 7.1 se muestra en línea roja cual sería la spline trazada para ir desde la salida de una sala hasta otra habitación al fondo de un pasillo, mientras que en color azul la que se generaría empleando únicamente rectas y segmentos de circunferencia.

El mayor problema que se podría encontrar en este planteamiento es que los cambios de uno a otro tipo de movimiento fuesen molestamente bruscos para el usuario, hecho que dependerá de la forma de programar el controlador de trayectorias y de las propias restricciones holonómicas del móvil.

- En cuanto al primero de los factores, se puede eliminar incluyendo en el controlador elementos como limitadores de aceleración, factores que dependen de la curvatura del giro, anchura del pasillo, longitud de un tramo, etc. Todos estos factores serán tenidos en cuenta a la hora de implementar el controlador de posición en el capítulo 8.

- En cuanto al segundo aspecto, hay que recordar que, tal y como se vio en el capítulo 2, la silla de ruedas es una plataforma con dos ruedas locas y dos que son motrices y directrices a la vez. Se trata de un sistema no holonómico, con lo que existen restricciones que disminuyen el espacio de movimiento a dos dimensiones. Sin embargo la posibilidad que incorpora la silla de girar sobre sí misma contrarresta esta discapacidad, empleando las tareas de giro tal y como se ha comentado.

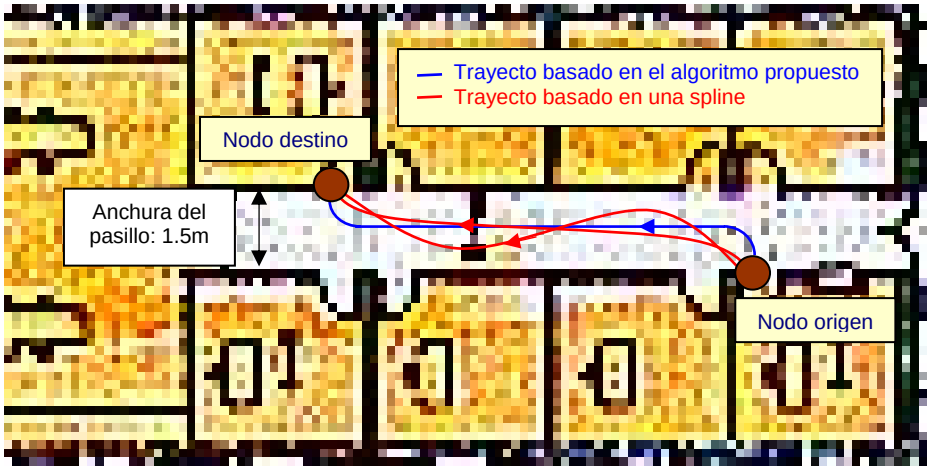


Figura 7.1. Comparativa entre curva spline y trayecto con rectas y arcos de circunferencia

Existen numerosos trabajos que emplean esta técnica con buenos resultados [7]. Estos trabajos se han movido en el mismos sentido que el aquí presentado: el de conseguir, con una adecuada fiabilidad, una conducción cómoda, en detrimento quizás de la exactitud del movimiento, que por otro lado es innecesaria muchas veces (cuando un ser humano recorre un pasillo no se plantea que ha de seguir una trayectoria concreta con un error de desviación menor de un %).

7.2. Algoritmo basado en tareas genérico.

Como ya se ha comentado, el generador de trayectorias es llamado por el gestor, que le indica cuál es la pareja de nodos que tiene que unir. Con esta información, el generador de trayectorias ha de dividir el tramo en varias tareas y obtener las consignas para que el controlador asegure el recorrido entre dos nodos. En este

sentido el generador de trayectorias se comporta más bien como un generador de tareas para el controlador.

Tal y como ya se ha comentado, el generador descompone cada uno de los tramos de la ruta (espacio entre dos nodos) en una secuencia de tareas de dos tipos:

a) *Operaciones de giro:* La trayectoria es un segmento de circunferencia de radio constante. Son generadas cuando el móvil ha de cambiar de dirección.

b) *Operaciones de avance.* La trayectoria es una simple recta. Permite avanzar por corredores hasta alcanzar un cruce o una sala.

La forma de realizar la descomposición del tramo en tareas está basada en el comportamiento de un usuario que se mueva con una silla de ruedas conducida de forma manual:

- El usuario conduce la silla de ruedas en línea recta a lo largo de los pasillos que encuentra en su recorrido.
- Al llegar al punto de acceso a una habitación o en el punto de inflexión o el cruce de dos pasillos y a una distancia corta, es cuando únicamente realiza cambios de dirección.

Si se aprovecha el hecho de que, a la hora de realizar el modelado del entorno en este trabajo, los nodos se han localizado justamente en las puertas de acceso y en los cruces de pasillos, se observa que en la mayoría de los casos los movimientos naturales de un usuario de una silla de ruedas se restringen a dos: el giro en las proximidades de los nodos (operación de giro) y el tramo de avance en línea recta entre los nodos propiamente dicho (operación de avance). Son justamente las trayectorias que permite generar el algoritmo de trayectorias.

7.2.1. Mapa del entorno para el generador de trayectorias.

Establecido lo anterior, es necesario fijar unos criterios que le permitan al generador diseñar los dos tipos de movimientos. El método más sencillo para discriminar las trayectorias será mediante información que le indique cuándo existe un cambio de dirección, y ya que no es posible conocerlo a través de la posición real de los nodos, es necesario incluir dos nuevos campos en el mapa.

1. *La dirección normal al nodo:* Informa de la dirección en que hay que afrontar el paso por ese nodo. Recordando que los nodos se sitúan en la parte superior de las puertas y accesos este dato facilita además los pasos a través de las puertas, una de las tareas más complejas en la navegación por interiores.

El sentido del ángulo de orientación normal depende del tipo de nodo que se trate. Como regla general, se toma el sentido de salida de la sala, cuando se trata de un nodo genérico (asociado a una sala) y el sentido de salida a un nivel de jerarquía superior cuando se trata de un nodo jerárquico. Este punto se tendrá que tener en consideración a la hora de desarrollar la algoritmia de generación de trayectorias, tal y como se verá más adelante.

2. *La distancia de aproximación a nodo*: Informa sobre la distancia en torno a la cual se van a realizar los giros asociados al nodo. Si bien este parámetro parece innecesario, pues podría ser constante en todo el movimiento, es, indirectamente, el que va a permitir adaptar el modo de conducción al entorno concreto del mapa donde se encuentre el móvil. Permite determinar el radio de giro, la velocidad de avance, etc., variables que deberán ser mayores o menores en función de la anchura del pasillo donde se encuentre el acceso, de la comodidad que se desea para el giro, del tramo siguiente a afrontar, etc. Todas estas consideraciones deberán ser tenidas en cuenta al fijar la distancia de aproximación al nodo, en el mapa.

En la figura 7.2 se muestra un arrea de la planta 2º del ala oeste del edificio bajo estudio, en la cual se han añadido los dos nuevos parámetros a los nodos.

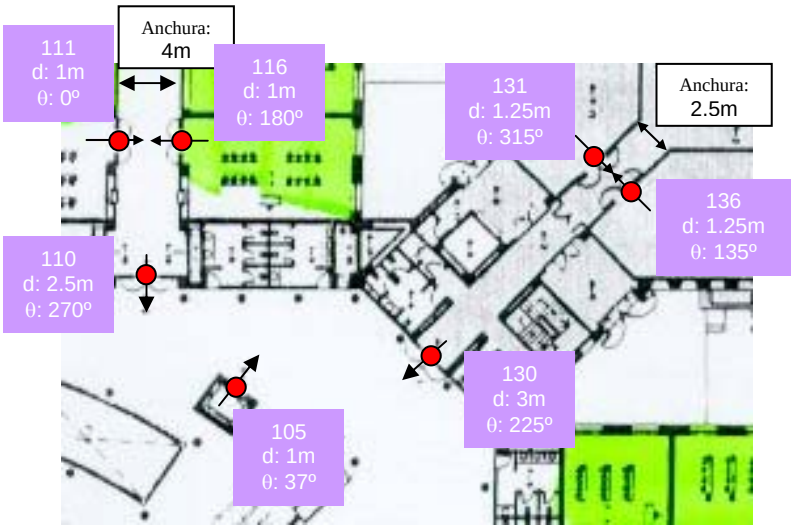


Figura 7.2. Ejemplos de dirección normal al nodo y distancia de aproximación al nodo en un segmento de mapa.

Tal y como se ha comentado, el valor de la distancia de aproximación se fija para hacer más cómodo el movimiento del robot, y en la figura 7.2 aparecen algunos casos que merece la pena analizar:

- Los pasillos de esta planta son bastante estrechos, dificultando por tanto la conducción de una silla de ruedas. En los nodos de estos pasillos la distancia de aproximación al nodo coincide con la mitad de la anchura del mismo, coincidiendo

con la localización de la puerta de acceso al pasillo. De este modo se consigue que el móvil circule justamente por el centro del pasillo.

- En los movimientos alrededor de la zona central de la planta, la distancia de aproximación al nodo aumenta considerablemente. Esto conlleva un aumento del radio de los giros que trazará la silla en sus movimientos, haciendo que la conducción sea más cómoda y aprovechando todo el espacio existente.

En la figura 7.2 aparece también un nodo de transición. Estos nodos no influyen el campo de dirección normal al nodo, pues no tiene sentido forzar al móvil en una dirección concreta al pasar por ellos, al no coincidir ni con puerta ni con ningún tipo de acceso especial. La distancia de aproximación en estos nodos es también grande, permitiendo describir arcos de mayor amplitud.

En cuanto a la dirección normal al nodo, se puede apreciar en la figura 7.2, que se determina como la orientación perpendicular a la determinada por la puerta o el acceso en que se encuentre el nodo, referenciada al sistema de coordenadas cartesianas establecido (ver punto 5.5).

Fijando este parámetro se asegura que la silla de ruedas atraviesa las puertas siempre en esa dirección, facilitando el acceso y evitando posibles choques con los marcos. De nuevo se ha deseado imitar la conducción manual del móvil, por pensar que es la más adecuada y dar confortabilidad al usuario.

La distancia de aproximación al nodo aparece, al igual que el par de coordenadas (x,y), en cm escalados, con escala 1:500, mientras que la orientación normal al nodo aparece en grados respecto al eje x del sistema cartesiano definido.

Establecidos los nuevos parámetros a incluir en el mapa, se muestra a continuación cual es el aspecto final del mismo. La leyenda sería la siguiente:

Nombre_nodo coordenada_x coordenada_y distancia_aproximación orientación_normal

Apariencia final del mapa del ala oeste del Edificio Politécnico.

```
000 0 0 180 0.3
001 -0.3 -1 90 0.2
002 3.7 -0.7 55 0.2
003 0.5 0.5 270 0.5
004 7 -6 180 0.5
005 5.7 -2.7 37 0.2
006 5.6 -6.7 0 0.2
007 6.5 -6.5 270 0.2
008 2.8 -3 45 0.2
010 2.5 0.5 270 0.5
011 2 3 0 0.2
012 2.2 5 0 0.2
013 3.2 5 180 0.2
014 3.3 3 0 180 0.2
020 7 -3.7 180 0.5
021 7 3.3 180 0.5
022 7 3.3 180 0.5
023 7 3.3 180 0.5
024 7 3.3 180 0.5
030 6 -0.5 225 0.6
031 9 2.7 315 0.25
032 9.5 2.4 135 0.25
033 10.5 4 225 0.5
-001 5.7 -0.8 0.5
```

```
NE
NE
NE
NE
100 0 0 180 0.3
101 -0.3 -1 90 0.2
102 3.7 -0.7 55 0.2
103 0.5 0.5 270 0.5
104 7 -6 180 0.5
105 5.7 -2.7 37 0.2
106 5.6 -6.7 0 0.2
107 6.5 -6.5 270 0.2
110 2.6 0.5 270 0.5
111 2.2 2.2 0 0.2
112 2.2 3.6 0 0.2
113 2.2 5 0 0.2
114 3 5 180 0.2
115 3 3.6 180 0.2
116 3 2.2 180 0.2
120 7 -3.8 180 0.4
121 9 -3.4 270 0.2
122 10.4 -3.4 270 0.2
123 11.8 -3.4 270 0.4
124 11.8 -4.2 90 0.4
125 10.4 -4.2 90 0.4
126 9 -4.2 90 0.4
130 6 -0.5 225 0.6
131 8.2 2 315 0.25
132 9 4 270 0.5
133 10.3 4.3 315 0.25
134 10.9 3.8 135 0.25
135 10.8 2.4 180 0.2
136 8.6 1.6 135 0.2
137 10.8 4.3 225 0.4
-101 5.7 -0.8 0.5
-131 9 2.4 0.2
200 0 0 180 0.3
201 -0.3 -1 90 0.2
202 3.7 -0.7 55 0.2
205 5.7 -2.7 37 0.2
206 5.6 -6.7 0 0.2
207 6.6 -6.5 270 0.2
210 1.15 0.4 270 0.4
211 1 1.8 0 0.15
212 1 2.2 0 0.15
213 1 2.9 0 0.15
214 1 3.6 0 0.15
215 1.3 3.3 180 0.15
216 1.3 2.7 180 0.15
217 1.3 2 180 0.15
218 1.3 1.5 180 0.15
219 1.15 3.9 270 0.3
220 4.15 0.4 270 0.4
221 4 1.5 0 0.15
222 4 2 0 0.15
223 4 2.7 0 0.15
224 4 3.3 0 0.15
225 4.3 3.6 180 0.15
226 4.3 2.9 180 0.15
227 4.3 2.2 180 0.15
228 4.3 1.8 180 0.15
229 4.15 3.9 270 0.3
230 7 -2.35 180 0.4
231 8.9 -2.2 270 0.15
232 9.3 -2.2 270 0.15
233 10.1 -2.2 270 0.15
234 10.8 -2.2 270 0.15
235 10.4 -2.5 90 0.15
236 9.8 -2.5 90 0.15
237 9.1 -2.5 90 0.15
238 8.1 -2.5 90 0.15
239 11.1 -2.35 180 0.4
240 7.1 -5.35 180 0.4
241 8.6 -5.2 270 0.15
242 9.1 -5.2 270 0.15
243 9.6 -5.2 270 0.15
244 10.4 -5.2 270 0.15
245 10.8 -5.5 90 0.15
246 10.1 -5.5 90 0.15
247 9.3 -5.5 90 0.15
248 8.9 -5.5 90 0.15
249 11.1 -5.35 180 0.35
250 6.1 -0.4 225 0.6
251 8.2 1.9 315 0.25
252 9 4 270 0.5
253 10.3 4.2 315 0.25
254 10.9 3.8 135 0.25
255 10.8 2.4 180 0.2
256 8.6 1.6 135 0.2
257 10.9 4.2 225 0.4
261 8.1 -2 270 0.35
262 8.1 -2.7 90 0.35
-201 5.7 -0.8 0.5
-251 9 2.4 0.2
NE
NE
300 0 0 180 0.3
301 -0.3 -1 270 0.2
302 2.65 -0.2 90 0.2
303 5.5 -0.7 45 0.35
304 6.4 -2 0 0.2
305 6.4 -3.9 0 0.2
306 5.6 -6.7 0 0.2
307 6.6 -6.5 270 0.2
310 1.15 0.35 270 0.35
311 1 1.8 0 0.15
312 1 2.2 0 0.15
313 1 2.9 0 0.15
314 1 3.6 0 0.15
315 1.3 1.5 180 0.15
316 1.3 2 180 0.15
317 1.3 2.7 180 0.15
318 1.3 3.3 180 0.15
319 1.15 3.9 270 0.3
320 4.15 0.35 270 0.35
321 4 1.5 0 0.15
322 4 2 0 0.15
323 4 2.7 0 0.15
324 4 3.3 0 0.15
325 4.3 3.6 180 0.15
326 4.3 2.9 180 0.15
327 4.3 2.2 180 0.15
328 4.3 1.8 180 0.15
329 4.15 3.9 270 0.3
330 7 -2.35 180 0.4
331 8.9 -2.2 270 0.15
332 9.3 -2.2 270 0.15
333 10.1 -2.2 270 0.15
334 10.8 -2.2 270 0.15
335 10.4 -2.5 90 0.15
336 9.8 -2.5  90 0.15
337 9.1 -2.5 90 0.15
338 8.1 -2.5 90 0.15
339 11.1 -2.35 180 0.4
340 7.0 -5.35 180 0.4
341 8.6 -5.2 270 0.15
```

```
342 9.1 -5.2 270 0.15
343 9.6 -5.2 270 0.15
344 10.4 -5.2 270 0.15
345 10.8 -5.5 90 0.15
346 10.1 -5.5 90 0.15
347 9.3 -5.5 90 0.15
348 8.9 -5.5 90 0.15
349 11.1 -5.35 180 0.15
361 8.1 -2 270 0.35
362 8.1 -2.7 90 0.35
-301 5.15 0 0.3
-302 6.6 -1.35 0.3
```

El mapa completo tiene 3135 bytes, incluyendo los espacios en blanco para separar los parámetros, lo cual es un tamaño aceptable teniendo en cuenta que es la cantidad de memoria a manejar por los algoritmos de navegación a la hora de realizar la búsqueda de parámetros. Es necesario recordar que un tamaño mayor ralentizaría en exceso los algoritmos de búsqueda, pudiendo incluso llegar a afectar al periodo de muestreo del control.

Además este valor permite calcular de forma aproximada el tiempo de transferencia del mapa vía radio desde los nodos transmisores. Suponiendo una velocidad de transferencia estándar de 19200bps (velocidad típica de un transceiver radio) el tiempo total de transferencia sería de 1.3s, lo cual es del todo inapreciable para el usuario que va en la silla. De nuevo un mayor tamaño requeriría más tiempo, necesitando incluso detener el móvil para transferir el mapa.

Por otro lado, la construcción del mapa de este modo no resulta complicada, pues los parámetros a incluir son pocos y sencillos de obtener de forma automática mediante un análisis con un sistema de visión de los planos del edificio. El único parámetro que requiere un estudio manual y algo más detallado es la distancia de aproximación al nodo, pero teniendo en cuenta que afecta directamente a la confortabilidad del movimiento, es quizás preferible que sea así.

7.2.2. Algoritmo de segmentación de los tramos en tareas.

Con la información contenida en el mapa, el algoritmo de generación de trayectorias ha de determinar cuál va a ser la secuencia de tareas a incluir para recorrer un tramo. Teniendo en cuenta que cada tramo une dos nodos que pueden ser una sala del edificio o un simple punto de acceso a un pasillo, la secuencia general de un tramo en tareas va a ser la siguiente:

1. *Tarea de giro inicial.* En caso de que sea necesaria para incorporarse a un pasillo al salir de una sala o de otro pasillo.
2. *Tarea de avance.* Permite unir las tareas de giro asociadas con los nodos de inicio y fin del tramo.
3. *Tarea de giro final.* En caso de que sea necesario girar para dejar el móvil en la dirección normal al nodo indicada por el mapa.

La forma del algoritmo completo se encuentra descrita en el diagrama de flujo de la figura 7.3.

La obtención de la información relacionada con el nodo se realiza siempre a través del nombre del nodo, pues es la única información con la que cuenta el generador de trayectorias, procedente del gestor.

Tal y como se observa en el diagrama de la figura 7.3, para determinar cuándo es necesario incluir cada una de las tareas de giro, el algoritmo de generación de trayectoria se basa en los llamados *"puntos de aproximación al nodo".* Estos puntos se obtienen a partir de los *"puntos de acceso al nodo",* obtenidos a su vez con los parámetros asociados al nodo que se recogen del mapa.

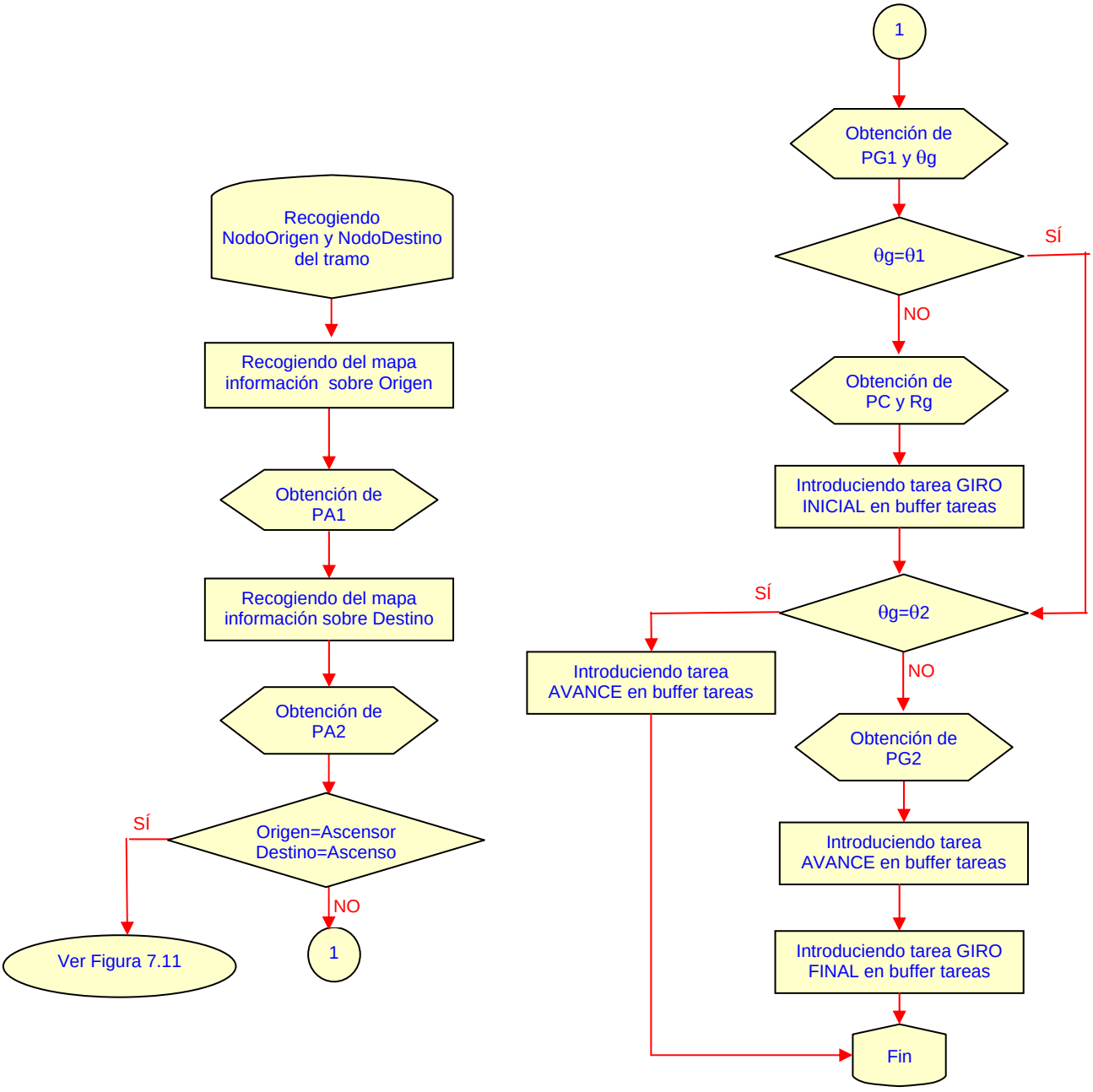


Figura 7.3. Diagrama de flujo general del algoritmo de generación de trayectorias.

El modo de obtener los puntos de acceso y los puntos de aproximación se detalla más adelante en la memoria. Pero dejando de un lado ese tema, el algoritmo de generación de trayectorias se apoya en ellos para determinar cuándo es necesario incluir cada una de las tareas de giro, para lo cual compara:

- *Para la tarea de giro inicial:* la dirección normal al nodo origen con la dirección del móvil en el punto de aproximación al nodo de origen.

- *Para la tarea de giro final:* la dirección normal al nodo destino con la dirección del móvil en el punto de aproximación al nodo de destino.

En caso de que sean iguales no será necesario incluir la tarea de giro asociada, y en caso contrario se obtendrán los parámetros que permiten trazar el giro. Estos parámetros se obtienen de los puntos de acceso al nodo, tal y como se explica más adelante.

La tarea de avance siempre se incluye en el tramo, si bien en caso de no ser necesaria el punto de origen y de fin de la misma coincidirán con lo que el robot no avanzará.

Como resumen de lo presentado, se muestra en la figura 7.4 la división en tareas de cada uno de los tramos contenidos en un trayecto completo tomado como ejemplo. Para distinguir las tareas se han macado con distintos colores: en azul se muestran las tareas de giro y en verde las de avance.

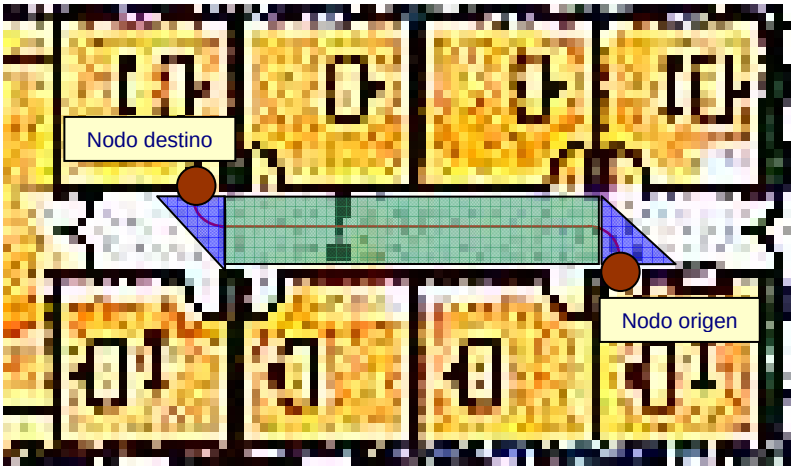


Figura 7.4. Ejemplo de división en tareas de un trayecto concreto.

En la figura se puede apreciar cómo la trayectoria en cada tramo se adapta al entorno en que se encuentre, gracias al parámetro de distancia de aproximación a nodo.

- En tramos como el recto que aparece en la figura 7.4, la silla avanza por el centro de un pasillo, realizando los accesos a los nodos del tramo mediante cambios de orientación de 90º con radios de giro pequeños

- En otros tramos se aprovecha más la anchura del corredor y los giros se realizan con mayor radio de curvatura, haciendo más suave la conducción al existir un mayor espacio de maniobra.

7.2.3. Algoritmo de generación de tarea de giro.

Como se ha comentado, la base para la generación de las trayectorias de giro está en la obtención de *los puntos de acceso y de los puntos de aproximación* al nodo.

En la figura 7.5 se muestra de forma gráfica la relación entre el nodo y estos dos puntos, cuya obtención y significado es el siguiente:

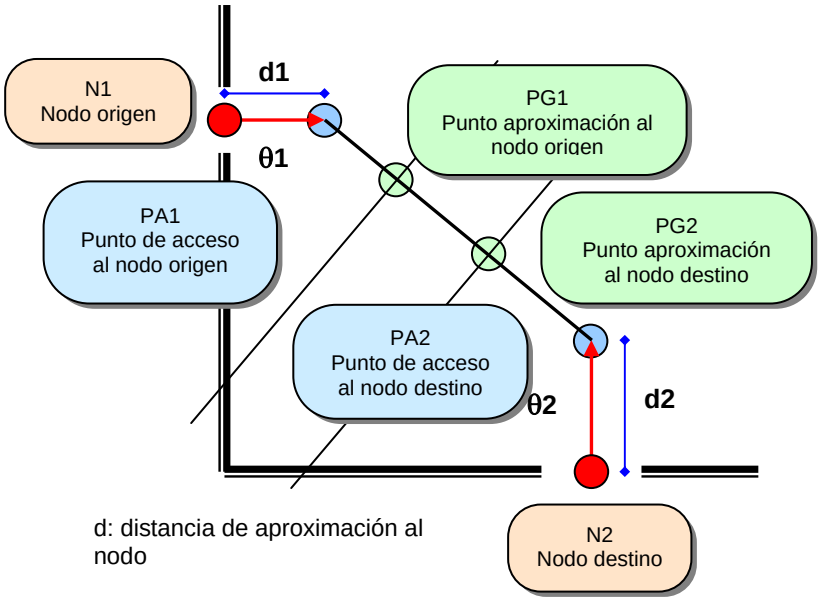


Figura 7.5. Gráfico de generación de los puntos de acceso, y de aproximación al nodo.

El significado de los parámetros que aparecen en el gráfico 7.5 se muestra en la tabla de la figura 7.6.

Parámetro	Identificación
N1 (x1,y1)	Nodo de origen del tramo
N2 (x2,y2)	Nodo de destino del tramo
d1	Distancia de aproximación al nodo origen
d2	Distancia de aproximación al nodo destino
θ1	Orientación normal al nodo origen
θ2	Orientación normal al nodo destino
PA1 (xa1,ya1)	Punto de acceso al nodo origen

PA2 (xa2,ya2)	Punto de acceso al nodo destino
PG1 (xg1,yg1)	Punto de aproximación al nodo origen
PG2 (xg2,yg2)	Punto de aproximación al nodo destino

Figura 7.6. Tabla de parámetros del cálculo de los puntos de acceso y aproximación a los nodos de origen y destino del tramo.

a) *Los puntos de acceso al nodo (PAi)* se localizan a una distancia igual a la de aproximación al nodo y en la dirección normal al nodo, con lo que se obtienen directamente a partir de la información asociada al nodo que se encuentra en el mapa. Se utilizan solo de forma auxiliar, y no forman parte de la trayectoria real del nodo.

   A la hora de calcular el punto de acceso al nodo destino, es importante tener en cuenta el sentido del ángulo de orientación al nodo, que, como ya se comentó en el apartado 7.2.1 anterior, depende del tipo de nodo. Por esto, antes de pasar al cálculo de las coordenadas en sí, el algoritmo realiza una identificación jerárquica del nodo para así añadirle el sentido correcto a la dirección de acceso al nodo. De este modo se obtiene la *orientación normal al nodo modificada (θ*),* que va a ser empleada varias veces a lo largo de todo el desarrollo del algoritmo de generación de trayectorias

   Una vez tenido en cuenta ese aspecto relacionado con la normal al nodo, las coordenadas del punto de acceso se obtienen mediante las ecuaciones 7.1 y 7.2, en función de la distancia de aproximación (d) y la orientación normal modificada ($\theta*$).

$$xai = xi + d \cdot \cos(\theta i^*) \qquad <7.1>$$

$$yai = yi + d \cdot sen(\theta i^*) \qquad <7.2>$$

   Donde el sufijo *'i'* será 1 si se trata del nodo origen y 2 cuando se refiera al nodo de destino.

b) *El punto de aproximación al nodo 'i' (PGi)* se obtiene a partir del punto de acceso al nodo 'i' y del punto de acceso al otro nodo que conforma el tramo. Constituyen el final de la tarea de giro o avance que se genere, por lo que la información de su posición será la que se proporcione al controlador de trayectorias.

La forma de obtener la localización de los nodos de aproximación se puede observar perfectamente en la figura 7.5:

- Se obtiene la ecuación de la recta que forman los puntos de acceso a los nodos origen y destino del tramo en cuestión.

- El punto de aproximación al nodo será aquél que pertenezca a la recta obtenida y se encuentre a una distancia igual a la distancia de aproximación del nodo, y en el segmento delimitado por los puntos de acceso a los nodos.

Las ecuaciones 7.3 y 7.4 permiten obtener las coordenadas de los puntos de aproximación al nodo. El sufijo *'i'* será 1 para obtener el punto de aproximación al nodo de origen y 2 para el punto de aproximación al nodo de destino del tramo.

$$ygi = yai + \frac{di}{\sqrt{\left(\frac{xa2 - xa1}{ya2 - ya1}\right)^2 + 1}} \qquad <7.3>$$

$$xgi = xai + \frac{xa2 - xa1}{ya2 - ya1} \cdot \left(ygi - yai\right) \qquad <7.4>$$

Una vez calculados estos puntos es posible determinar otros parámetros que permiten caracterizar la trayectoria de giro: *el radio de curvatura del giro, la dirección de salida de la tarea de giro y el centro de la circunferencia* a la que pertenece el arco de giro.

Evidentemente, será necesario calcular estos parámetros tanto para la tarea de giro de salida del nodo origen como para la tarea de giro de entrada al nodo de destino. Para facilitar la nomenclatura no se añadirá la terminación '*i*', que indicaría como siempre, si se trata de parámetros característicos de la tarea de giro en origen (sería sufijo 1) o del giro en el nodo de destino (sería sufijo 2).

En la figura 7.7 se observan de forma gráfica ambos parámetros, y en la tabla de la figura 7.8 se muestra la nomenclatura empleada para cada uno de ellos.

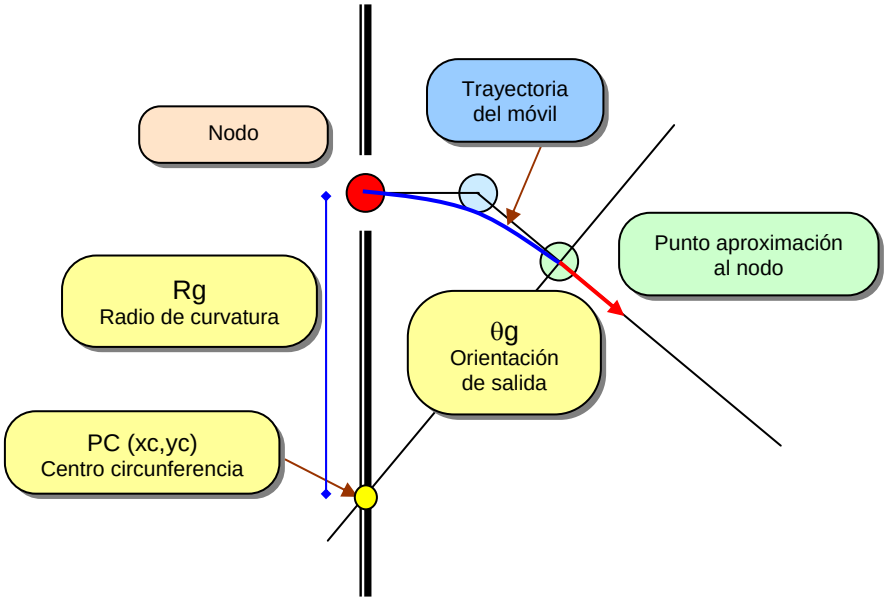


Figura 7.7. Obtención del radio y origen de curvatura y de la dirección de salida de la tarea de giro para el caso del nodo origen.

Parámetro	Identificación
θg	Orientación de salida de la tarea de giro
Rg	Radio de curvatura de la tarea de giro
PC (xc,yc)	Punto centro de la circunferencia de giro

Figura 7.8. Tabla de identificación de parámetros necesarios para la tarea de giro.

1. *La dirección de salida de la tarea de giro (θg)* se obtiene directamente como la pendiente de la recta que une los puntos de acceso a los nodos de origen y destino. La ecuación 7.5 muestra el procedimiento de cálculo.

$$\theta g = \arctan \frac{ya2 - ya1}{xa2 - xa1} \qquad <7.5>$$

2. *El centro de la circunferencia del arco de giro (PC)* se obtiene del cruce de dos rectas (ver figura 7.7):

   - La recta que pasa por el nodo con dirección perpendicular a la dirección normal al nodo (θi).
   - La recta que pasa por el punto de aproximación al nodo con dirección perpendicular a la dirección de salida de la tarea de giro (θg).

   En las ecuaciones 7.6 y 7.7 se presenta el método de cálculo de las coordenadas de este punto.

$$xc = \frac{y - \tan\left(\theta + \frac{\pi}{2}\right) \cdot x - yg + \tan\left(\theta g + \frac{\pi}{2}\right) \cdot xg}{\tan\left(\theta g + \frac{\pi}{2}\right) \cdot x - \tan\left(\theta + \frac{\pi}{2}\right) \cdot x} \qquad <7.6>$$

$$yc = \tan\left(\theta + \frac{\pi}{2}\right) \cdot (xc - x) + y \qquad <7.7>$$

3. *El radio de curvatura de la tarea de giro (Rg)* se obtiene como el radio de la circunferencia tangente a las dos rectas de inicio y fin de la tarea de giro:

   La ecuación 7.8 permite obtener el valor de este parámetro a partir del resto ya conocidos.

$$Rg = \sqrt{(x - xc)^2 + (y - yc)^2} \qquad <7.8>$$

Como se observa en la ecuación 7.8, el radio de curvatura queda determinado por la distancia de aproximación al nodo, pues ésta define a su vez la posición de los puntos de aproximación al nodo. Por ello, será necesario tener en cuenta esta dependencia a la hora de definir la distancia de aproximación al crear el mapa, como ya se había comentado anteriormente.

Tal y como se vio en el punto 7.2.2. de este capítulo, el generador de trayectorias obtiene en primer lugar los puntos de acceso a los nodos de origen y destino, y con ellos la dirección de salida de la tarea de giro inicial. Este parámetro le permite decidir (ver figura 7.3) si es necesario incorporar una tarea de giro, en cuyo caso realiza el resto de cálculos explicados en este apartado.

Tras la tarea de giro inicial, el generador de trayectorias incorpora una tarea de avance, y vuelve a realizar el mismo proceso para incorporar la tarea de giro final en caso necesario.

En la figura 7.9 se muestra el resultado de la obtención de los parámetros resultantes de aplicar el algoritmo de generación de trayectorias en un recorrido de ejemplo. Loa parámetros incluidos son los que posteriormente va a necesitar el controlador. La nomenclatura seguida es la misma se ha venido empleando en la descripción anterior (ver figuras 7.6 y 7.8).

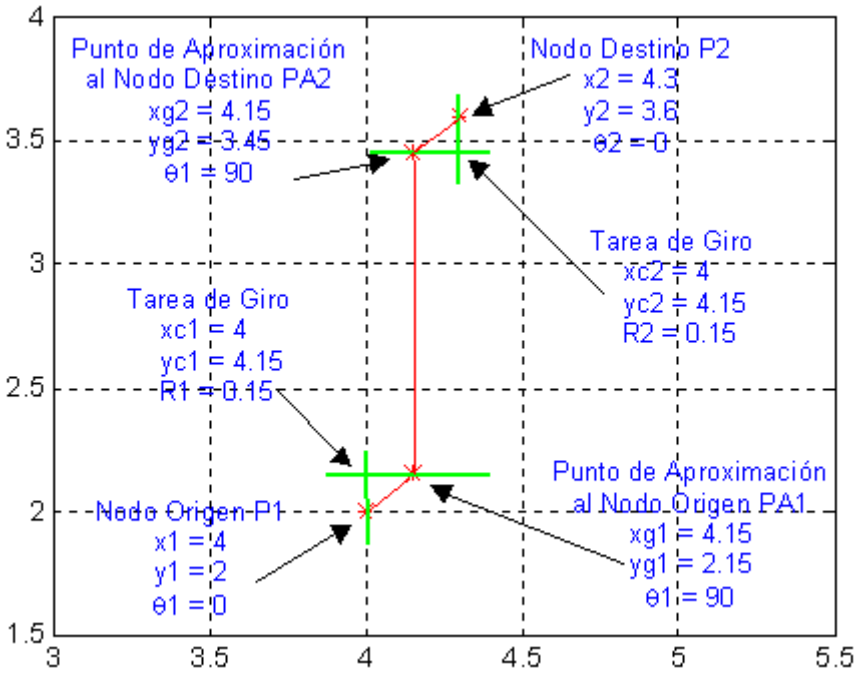


Figura 7.9. Resultado de la aplicación del algoritmo de giro a un tramo de ejemplo.

7.2.4. Parámetros de salida del algoritmo de generación de trayectorias.

Una vez realizado el cálculo de todos los parámetros que describen las dos tareas de giro, el generador de trayectorias recoge los valores adecuados que permiten describir cada una de las tareas y los concatena, dando lugar a una lista de variables que describen la trayectoria completa.

La relación entre los parámetros que describen la trayectoria y los valores calculados en el algoritmo de la tarea de giro depende de la composición de la trayectoria, que, tal y como se vio en el apartado 7.2.1., es función del tipo y de la localización de los nodos que compongan el tramo a generar.

En la tabla de la figura 7.10 se muestra cuales son los valores que describen cada una de las tareas implementadas por el generador, a partir de los parámetros obtenidos en el desarrollo del algoritmo. En la tabla se emplea la misma nomenclatura que en apartados anteriores, para poder así identificar fácilmente cada parámetro.

Tipo de Tarea	Condiciones	Parámetro del algoritmo asociado	Identificación
TAREA GIRO INICIAL		x1,y1	Coordenadas del punto inicial de la tarea
		$\theta 1^*$	Orientación de entrada en la tarea
		xg1,yg1	Coordenadas del punto final de la tarea
		θg	Orientación de salida de la tarea
		Rg	Radio de curvatura del giro
TAREA DE AVANCE	Si no existe tarea de giro inicial pero sí final	x1,y1	Coordenadas del punto inicial de la tarea
		xg1,yg1	Coordenadas del punto final de la tarea
		$\theta 1^*$ o θg	Orientación de entrada y salida de la tarea
TAREA DE AVANCE	Si existe tarea giro de giro inicial pero no final	xg1,yg1	Coordenadas del punto inicial de la tarea
		x2,y2	Coordenadas del punto final de la tarea
		θg o $\theta 2^*$	Orientación de entrada y salida de la tarea
TAREA DE AVANCE	Si no existe tarea de giro inicial ni final	x1,y1	Coordenadas del punto inicial de la tarea
		x2,y2	Coordenadas del punto final de la tarea
		$\theta 1^*$ o $\theta 2^*$	Orientación de entrada y salida de la tarea
TAREA DE AVANCE	Si existe tarea giro de giro inicial y final	xg1,yg1	Coordenadas del punto inicial de la tarea
		xg2,yg2	Coordenadas del punto final de la tarea
		θg	Orientación de entrada y salida de la tarea

TAREA GIRO FINAL		xg2,yg2	Coordenadas del punto inicial de la tarea
		θg	Orientación de entrada en la tarea
		x2,y2	Coordenadas del punto final de la tarea
		θ2*	Orientación de salida de la tarea
		Rg	Radio de curvatura del giro

Figura 7.10. Tabla de parámetros de distintas trayectorias

7.3. Algoritmo basado en tareas para tramos de cambio de planta.

Cuando el camino descrito por el planificador de rutas incluye un cambio de planta (ver figura 6.2) aparecen en la ruta dos nodos asociados a un ascensor: uno en la planta por la que se accede al ascensor y otro en la planta en la que se sale del ascensor. Los tramos que tiene sólo como origen o destino un nodo ascensor se trazan del mismo modo que cualquier otro, y llevan al móvil sólo hasta el umbral del ascensor.

Sin embargo la tarea que aparece entre dos nodos asociados a ascensores requieren un algoritmo especial, cuyo flujograma se muestra en la figura 7.11.

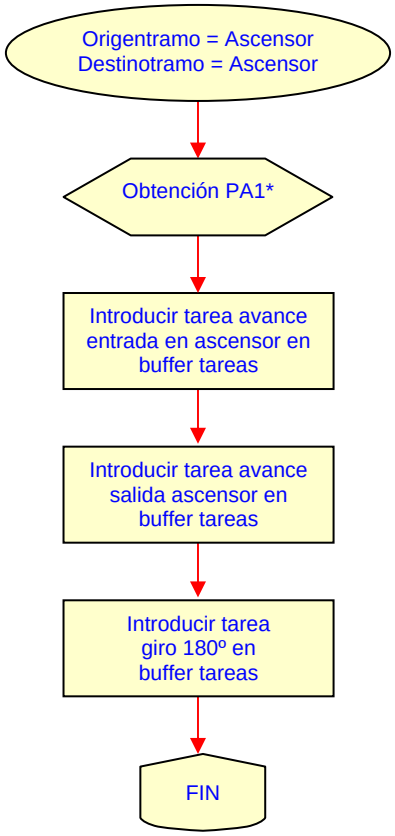


Figura 7.11. Algoritmo de generación de trayectoria en un cambio de planta

Como se observa en el diagrama, el tramo incluye una tarea de entrada en el ascensor (tarea de avance), otra de salida (de nuevo, tarea de avance) y una última de rotación de la silla 180º para que quede orientada de nuevo en sentido opuesto al ascensor.

Evidentemente, entre las dos tareas de avance el usuario ascensor debe transportar al usuario a la planta de destino, pero será tarea del controlador detectar esta situación y provocar la parada del móvil hasta entonces.

Sería interesante contar con un hardware que interactúa con el ascensor, para hacer la tarea de cambio de planta totalmente automática. Este aspecto no ha sido desarrollado en el proyecto y, si bien la tarea de generación de trayectorias queda cerrada en cuanto al cambio de planta se refiere, sería necesario implementar una función para la gestión de cambio de planta en el controlador

Es necesario añadir en este punto que, si bien el algoritmo de generación de trayectorias está perfectamente finalizado y simulado su funcionamiento dista de ser el óptimo, pues por la falta de un sistema de posicionamiento absoluto más fiable que el existente no se han podido realizar pruebas en campo con un usuario real.

Debido a que la trayectoria que aparece entre dos nodos asociados a ascensores es bastante particular, los parámetros que definen cada una de las tareas que la forman también lo son. En la tabla de la figura 7.12 se muestran, del mismo modo que se hizo en la figura 7.10, los parámetros que definen cada una de las tareas que conforman este tipo de trayectorias. En este caso, sin embargo, la secuencia de tareas es siempre la misma, por lo que la tabla queda bastante simplificada.

Tipo de Tarea	Parámetro del Algoritmo asociado	Identificación
TAREA DE AVANCE ACCESO ASCENSOR	x1,y1	Coordenadas del punto inicial de la tarea
	xa1*,ya1*	Coordenadas del punto final de la tarea
	θ1 o θ2 +180º	Orientación de entrada y salida de la tarea
TAREA DE AVANCE SALIDA ASCENSOR	xa1*,ya1*	Coordenadas del punto inicial de la tarea
	x2,y2	Coordenadas del punto final de la tarea
	θ1 o θ2 +180	Orientación de entrada y salida de la tarea
TAREA DE GIRO 180º	x2,y2	Coordenadas del punto inicial de la tarea
	θ1 o θ2 +180	Orientación de entrada de la tarea
	x2,y2	Coordenadas del punto final de la tarea
	θ1 o θ2	Orientación de salida de la tarea
	0	Radio de curvatura del giro

Figura 7.12. Tabla de parámetros de las tareas de un tramo entre ascensores

Las coordenadas xa1*,ya1* pertenecen al punto de acceso PA1*, que se calcula del mismo modo que se hace con PA1 (ver apartado 7.2.2) pero eligiendo como sentido del ángulo normal al nodo el de acceso al ascensor (θi+180 en grados). Las ecuaciones 7.1 y 7.2 mostraban el método de cálculo de estos valores.

7.4. Organización de resultados en la generación de trayectorias.

Una vez desarrollado todo el cálculo de generación de trayectorias, es necesario determinar y ordenar los parámetros que serán devueltos al gestor de procesos, como respuesta a la petición realizada, y para que el controlador de posición supervise la realización de la trayectoria.

Tal y como se explica en el capítulo 5 de la memoria, el paso de parámetros entre el generador y el controlador no está realmente supervisado por el gestor de procesos, sino que se realiza a través de un par de buffers circulares controlados por sendos flags de sincronismo.

En la tabla de la figura 7.13 se muestran los datos que necesita el controlador para poder implementar las trayectorias de avance y giro correspondientes a cada tarea. Los valores de estas variables se pueden obtener de las tablas de las figuras 7.10 y 7.12, en función de las características concretas de cada trayecto. En esencia será necesario proporcionar el tipo de trayectoria de que se trata así como las coordenadas del punto de origen y de fin de la tarea y la orientación del móvil a la entrada y la salida de la misma. Además, será necesario incorporar el valor del radio del arco descrito y las coordenadas del centro de la circunferencia de giro (que solo serán efectivos en el caso de las tareas de giro) pues serán de gran utilidad para la algoritmia desarrollada por el controlador (ponderación de las constantes del controlador, obtención de la consigna de posición, etc.).

Parámetro	Identificación
xi,yi	Coordenadas del punto inicial
θi	Orientación de entrada de la tarea
xf,yf	Coordenadas del punto final
θf	Orientación de salida de la tarea
R	Radio de curvatura del giro
xc,yc	Coordenadas del centro de la circunferencia del radio de giro
TIPO	Parámetro de identificación de la tarea

Figura 7.13. Tabla de parámetros resultado de la generación de trayectorias.

8. Control de Posición

Finalmente, el proceso de control es el encargado de obtener las consignas de velocidad necesarias para que el móvil describa las trayectorias diseñadas por el generador de trayectorias, minimizando el error de posición.

El control es de posición, pues las consignas son de esa clase. A parte de esta característica, el tipo de controlador puede ser muy variado (clásico, borroso, óptimo, adaptativo, robusto, etc.). La idea está en encontrar aquel que, dentro de ser más o menos sencillo, responda correctamente (en cuanto a velocidad de respuesta, amortiguamiento, etc.) a la dinámica de la silla de ruedas.

Existen diversos trabajos (ver capítulo 2) en los que se ha hecho un estudio más exhaustivo sobre el controlador de lo que aquí se pretende. En este trabajo no se pretende dar la solución óptima al control de posición, pero sí adaptarlo a las características concretas de la aplicación, de forma que, se consiga una conducción confortable y dependiente de las características puntuales del entorno.

8.1. Visión general de lazo de control.

Como se ha comentado, la función del controlador de posición es la de obtener las consignas de excitación para las ruedas motrices izquierda y derecha (ω_I y ω_D) que permitan seguir con el mínimo error posible la trayectoria deseada.

El controlador es la única tarea de todo el sistema de navegación que se ejecuta de forma periódica. El gestor (ver capítulo 5) es un proceso de ejecución cíclica, que va controlado cuando tiene que hacer una llamada al controlador, y el resto de procesos realizan su tarea en los huecos de ejecución entre dos llamadas consecutivas al controlador y, si es posible, en un solo tramo de la línea de proceso.

En la figura 8.1 se muestran las tareas llevadas a cabo por el proceso de control en cada período de muestreo, además de otras que no pertenecen a este proceso concreto, y cuya funcionalidad se describe a continuación.

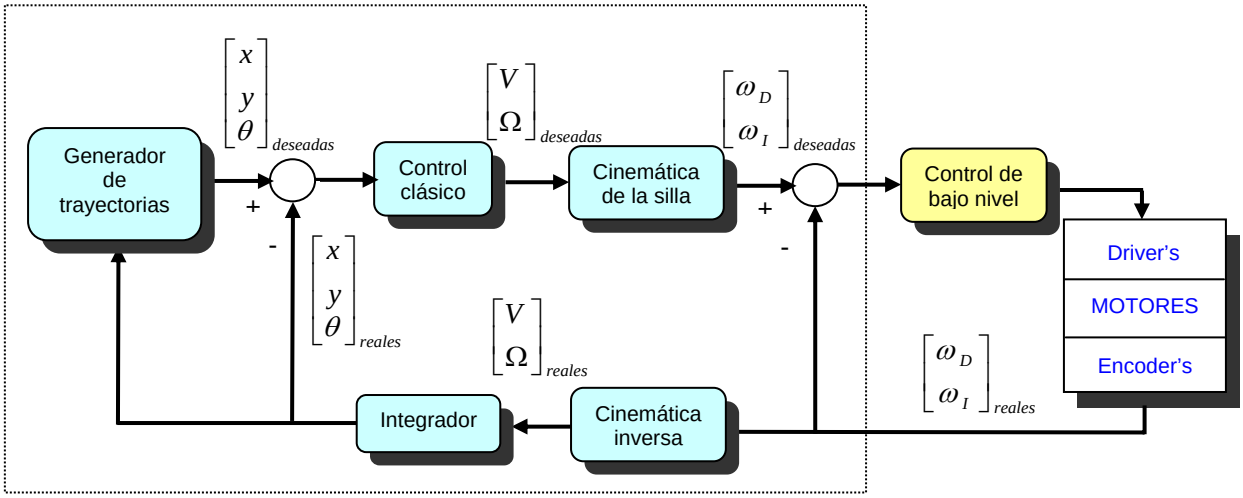


Figura 8.1. Diagrama de bloques del lazo de control.

a) *La posición real (xr,yr,θr)*. Para poder obtener el error de posición, a partir del cual se implementa todo el lazo de control, es necesario contar con la posición real y la consigna del móvil en cada período. La posición real se obtiene a partir de un sistema de posicionamiento absoluto, que en este caso parte de los encoders ópticos acoplados a las ruedas motrices. Es lo que se conoce como técnica deadreckoning.

b) *La consigna (xd,yd,θd).* A partir de los parámetros característicos de la trayectorias a seguir, proporcionados por el generados de trayectorias, el controlador debe extraer cuál es la consigna de posición adecuada para cada instante de muestreo. Tal y como se observa en la imagen, esta consigna depende de la posición real del móvil, práctica habitual en todos los sistemas de navegación.

c) *El error y el controlador*. Comparando las variables anteriores se obtiene el error de posición en los términos adecuados para aplicar después el algoritmo de control que permita obtener las consignas de velocidad para las ruedas del móvil bajo nivel. Se puede observar que existen dos controladores, uno para la velocidad de

avance (V) y otro para la velocidad de giro (Ω), que son además de muy distinto tipo, tal y como se verá más adelante.

d) *El bajo nivel*. Finalmente el resultado del lazo de control son las consignas de velocidad angular de las ruedas del móvil que sirven de consigna para el algoritmo de control de bajo nivel. Este algoritmo no ha sido desarrollado en este proyecto, pero sí ha sido adaptado para esta aplicación permitiendo que las consignas de velocidad angular procedentes del navegador se sigan con el mínimo error, mediante un nuevo lazo de control. En esta tarea será necesario también gestionar la interfaz que permite comunicar al PC, donde se realiza todo el algoritmo, con los controladores de bajo nivel, tal y como ya se ha comentado en capítulos anteriores.

La secuencia de ejecución de estas tareas de control se muestra en el diagrama de la figura 8.2.

La organización de las tareas viene fijada más por el propio encadenamiento de las mismas que por una razón de ingeniería. Existe, sin embargo, un significado real en el punto de ejecución de la tarea de deadreckoning, pues al ejecutarse en primer lugar en cada período de muestreo, se minimiza el jitter del período de repetición que puede aparecer si se realiza en último lugar, debido a la duración irregular del algoritmo de control.

Si bien cabría pensar que sucede lo mismo con el momento en que se proporcionan las consignas al bajo nivel, como resultado del control, no es así debido a la asincronía del lazo de control de bajo nivel.

A lo largo de todo el capítulo se realiza una descripción detallada de cada una de las tareas del controlador, así como de los parámetros necesarios para realizar un control que permita obtener los resultados deseados, a costa de particularizar las técnicas de control a la aplicación concreta, tal y como se ha venido haciendo con todos los procesos de navegación.

8.2. El periodo de repetición del algoritmo de control.

Como ya ha sido explicado, el proceso de control es el único proceso periódico de todo el sistema de navegación. La elección del periodo de muestreo es una de las tareas más complicadas del control, pues se encuentra limitado por dos elementos:

a) *La propia algoritmia desarrollada,* pues el período ha de ser suficientemente grande como para dar cabida a todas las tareas que realiza el controlador.

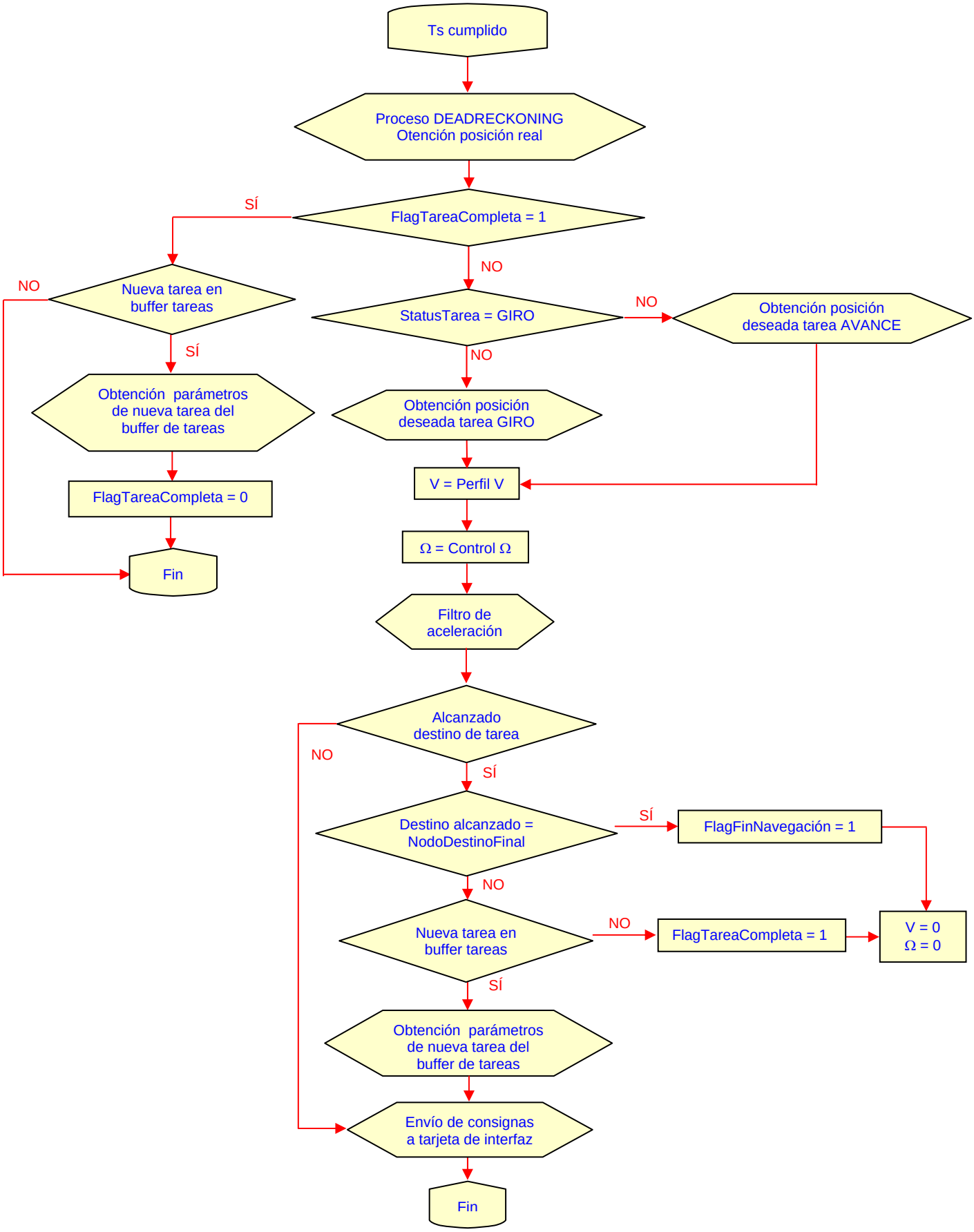


Figura 8.2. Diagrama de la secuencia de las tareas del lazo de control.

b) *El periodo de muestreo del sistema de bajo nivel,* pues no tendría sentido que el control de alto nivel fuese más rápido que el de bajo nivel que no podría en este caso reaccionar a los cambios indicados por este controlador de posición.

c) *La dinámica del sistema controlado,* pues en el proceso de deadreckoning, el periodo de muestreo debe ser suficientemente pequeño como para poder reconstruir el comportamiento del sistema continuo.

Estos tres elementos determinan el valor del periodo de muestreo. Dos de ellos vienen impuestos con independencia del algoritmo de control desarrollado:

- El sistema de control de bajo nivel tiene un periodo de repetición de 14ms, ajustado inicialmente en función de los parámetros del lazo de control implementado en este nivel [5], y retocado en este proyecto para evitar errores de jitter en el mismo (ver capítulo 9).

- Por otro lado, tal y como se comentaba en el capítulo 2, periodos de muestreo por debajo de los 200ms permiten al sistema discretizado comportarse prácticamente del mismo modo que el continuo.

Finalmente, de las medidas de temporización realizadas sobre el sistema se obtiene que el tiempo de ejecución del algoritmo de control implementado es de unos 30ms, por lo que, con los tres datos presentes, el periodo de repetición del algoritmo de control se puede fijar entre los 30ms y los 200ms.

Como solución de compromiso se ha resuelto fijar este tiempo a 50ms, valor algo mayor que el mínimo y muy inferior al máximo, con lo que se agiliza el sistema completo.

8.3. El sistema de posicionamiento. Deadreckoning.

Todo proceso de movimiento que incluye un control en lazo cerrado requiere un sistema de realimentación que le informe del estado del móvil para poder compararlo con la consigna y construir, a partir del error obtenido, la actuación correspondiente.

Uno de los sistemas de realimentación de posición más empleados son los encoders ópticos incrementales, que, acoplados al eje del motor y mediante un hardware adecuado para tratar las señales en cuadratura que proporciona, permite obtener información sobre el movimiento realizado por cada rueda del móvil en cada instante.

A partir de esta información se puede obtener la posición x, y, θ mediante un cambio de espacio de representación y empleando las ecuaciones de Fresnel. Es lo que se conoce como técnicas de *deadreckoning*.

El mayor inconveniente de estas técnicas está en la inexactitud del deadreckoning debido a deslizamientos de las ruedas motrices, que hacen incrementar el contador de pulsos en cuadratura sin que el móvil cambie de posición. Además, estos errores son acumulados, por lo que pueden llegar a ser importantes.

Es por ello que en la mayor parte de los casos [1] se incluyen, a parte de este o en vez de este, otros métodos de obtener la posición real del móvil. En el caso que aquí se trata, tal y como se planteó en capítulos anteriores, la idea es apoyar el deadreckoning con un sistema de visión, que proporciona la posición absoluta del móvil mediante las marcas artificiales y el mapa del entorno.

El emplear el sistema de visión como único método de realimentación no suele dar buenos resultados, ya que los algoritmos de procesamiento de imagen suelen ser bastante lentos debido a que la cantidad de información a tratar es bastante grande. Esto implica que el período de muestreo ha de crecer por encima de lo deseado y el sistema completo se vuelve lento.

En el trabajo presentado, el controlador emplea como información de realimentación de posición la que llega de los encoders en cada período de muestreo, y el sistema de visión solamente se usa para corregir los posibles errores del deadreckoning cada vez que se alcanza una marca, con lo que el error deja de ser acumulativo.

Como el sistema de visión no se ha implementado, hay que contar con los errores del deadreckoning. Sin embargo, estudios completos sobre el tema [7] muestran que solamente en casos de cambios bruscos de aceleración aparecen estos problemas, por lo que en el diseño del controlador se tendrá especial cuidado en el perfil de aceleración de las velocidades angulares de las ruedas, para evitar dichos errores.

Antes de pasar a la algoritmia del deadreckoning, es necesario plantear de qué modo se va a obtener la información sobre la velocidad de las ruedas, por lo que este punto se divide en dos apartados.

8.3.1. El acceso al bajo nivel.

Las señales en cuadratura procedentes del encoder son procesadas en el controlador de bajo nivel para proporcionar al navegador la acumulación de pulsos desde el inicio del movimiento. Esta información llega al PC a través de la interfaz con el bajo nivel, y se acompaña de una marca temporal que indica en qué instante se han hecho las medidas de pulsos.

En la tabla de la figura 8.3 se muestra la estructura de información que recibe el navegador desde el bajo nivel. Mencionar que sólo se muestra la estructura a nivel funcional, pues la forma de operar con la tarjeta de interfaz será explicada en el capítulo 9.

Parámetro	Identificación
T_D	Acumulación de tiempo de la rueda derecha
T_I	Acumulación de tiempo de la rueda izquierda
P_D	Acumulación de pulsos de la rueda derecha
P_I	Acumulación de pulsos de la rueda izquierda

Figura 8.3. Tabla con descripción funcional de los parámetros pasados desde el bajo nivel para obtener la velocidad real de la rueda.

A partir de estos valores es inmediato obtener la velocidad angular de cada rueda en mrad/s (miliradianes por segundo), mediante las ecuaciones 8.1 y 8.2.

$$\omega_D = \frac{P_D[n] - P_D[n-1]}{T_D[n] - T_D[n-1]} \cdot K_{CONV} \qquad [mrad/s] \qquad <8.1>$$

$$\omega_I = \frac{P_I[n] - P_I[n-1]}{T_I[n] - T_I[n-1]} \cdot K_{CONV} \qquad [mrad/s] \qquad <8.2>$$

En las ecuaciones anteriores se ha empleado la misma nomenclatura que en la tabla de la figura 8.3, indicando cuándo se trata de la muestra obtenida en ese período de muestreo ([n]) o en el anterior ([n-1]). Además ha sido necesario añadir una constante que permite realizar una conversión de magnitudes, teniendo en cuenta las características físicas del sistema de encoders. El valor de esta constante es:

$$K_{CONV} = \frac{2 \cdot \pi}{N_P \cdot N_R \cdot Tps} = \frac{2 \cdot \pi \cdot 1000}{400 \cdot 32 \cdot 0.82} = 598.626 \qquad <8.3>$$

donde:

- N_P*:* es el número de pulsos por vuelta que proporcionan los encoders
- N_R*:* es la relación de reductora del motor.
- *Tps:* es la constante que permite pasar a segundos la información temporal proporcionada por el bajo nivel.

La obtención de la velocidad se realiza en el propio navegador en vez de en el bajo nivel, para obtener una mayor resolución en la medida, y por la mayor velocidad de proceso del PC.

8.3.2. El deadreckoning.

Obtenida la información de posición incremental es posible calcular la terna *x*, *y*, θ aplicando las ecuaciones 8.4, 8.5 y 8.6, que permiten integrar el incremento de posición. Son las tres ecuaciones de Fresnel que constituyen la técnica del deadreckoning.

$$\theta[n] = \theta[n-1] + \Omega[n] \qquad [rad] \qquad <8.4>$$

$$x[n] = x[n-1] + V[n] \cdot \cos(\theta[n]) \cdot T_m[n] \qquad [m] \qquad <8.5>$$

$$y[n] = y[n-1] + V[n] \cdot sen(\theta[n]) \cdot T_m[n] \qquad [m] \qquad <8.6>$$

En las ecuaciones anteriores aparece la variable T_m. Se trata del valor medio del incremento de tiempo existente entre dos lecturas, medidos en la rueda derecha y la rueda izquierda (ec. 8.7). Se podría pensar que este tiempo coincide con el período de muestreo del sistema, pero no es así, pues, como ya se ha repetido varias veces, el proceso de navegación (y con el controlador) es asíncrono al bajo nivel, y tienen periodos de muestreo distintos. En el capítulo 6 se observa un estudio más detallado acerca de este tema.

$$T_m[n] = \frac{T_D[n] + T_I[n]}{2} \cdot Tps \qquad [s] \qquad <8.7>$$

Para poder aplicar las ecuaciones del deadreckoning, es necesario emplear primero las ecuaciones de la cinemática inversa de la silla de ruedas, para obtener la velocidad de avance (V) y de giro (Ω) asociadas a las velocidades angulares de las ruedas proporcionadas por el bajo nivel. Dichas ecuaciones (ver capítulo 2) son las que se muestran a continuación (8.8 y 8.9).

$$V[n] = \frac{\omega_D[n] + \omega_I[n]}{2} \cdot R \qquad [m/s] \qquad <8.8>$$

$$\Omega[n] = \frac{\omega_D[n] - \omega_I[n]}{D} \cdot R \qquad [rad/s] \qquad <8.9>$$

Donde *D* es la distancia entre ruedas del móvil (56cm) y *R* es el radio de la rueda (17cm).

A la hora de trabajar con esta parte es muy importante tener en cuenta la precisión de cálculo, pues trabajando con períodos de muestreo pequeños las lecturas del encoder van a ser pequeñas (de ahí el trabajar con mrad/s en vez de rad/s o rpm) con lo que un pequeño error de precisión en el deadreckoning aumentaría el porcentaje de error del sistema de posicionamiento absoluto.

8.4. La consigna de posición.

El otro punto de partida que permite obtener el error sobre el que se construye el sistema de control es la consigna de posición. Este dato indica al controlador dónde se debería encontrar el móvil dentro de la trayectoria marcada por el generador de trayectorias.

Para obtener la consigna de posición, por tanto, el controlador hace uso de la información recibida del generador de trayectorias, que, en forma de parámetros, definen la forma de la trayectoria. Dichos parámetros se vuelven a presentar en la siguiente figura 8.4, para facilitar la lectura de la memoria.

Parámetro	Identificación
xi,yi	Coordenadas del punto inicial
θi	Orientación de entrada de la tarea
xf,yf	Coordenadas del punto final
θf	Orientación de salida de la tarea
R	Radio de curvatura del giro
xc,yc	Coordenadas del centro de la circunferencia del arco de giro
TIPO	Parámetro de identificación de la tarea

Figura 8.4. Tabla de parámetros procedente del generador de trayectorias que permiten definir la trayectoria a seguir por el móvil

Además de información sobre la trayectoria a diseñar, el controlador necesita la posición actual del móvil para obtener la consigna de posición más adecuada. Es decir, el generador de trayectorias elegirá como consigna aquel punto que cumpla las tres premisas siguientes (ver figura 8.5):

1. Ha de ser el más próximo al punto real en que se encuentre el móvil.
2. Ha de pertenecer a la trayectoria que el generador indique.
3. Ha de encontrarse en el sentido de avance que el móvil está siguiendo.

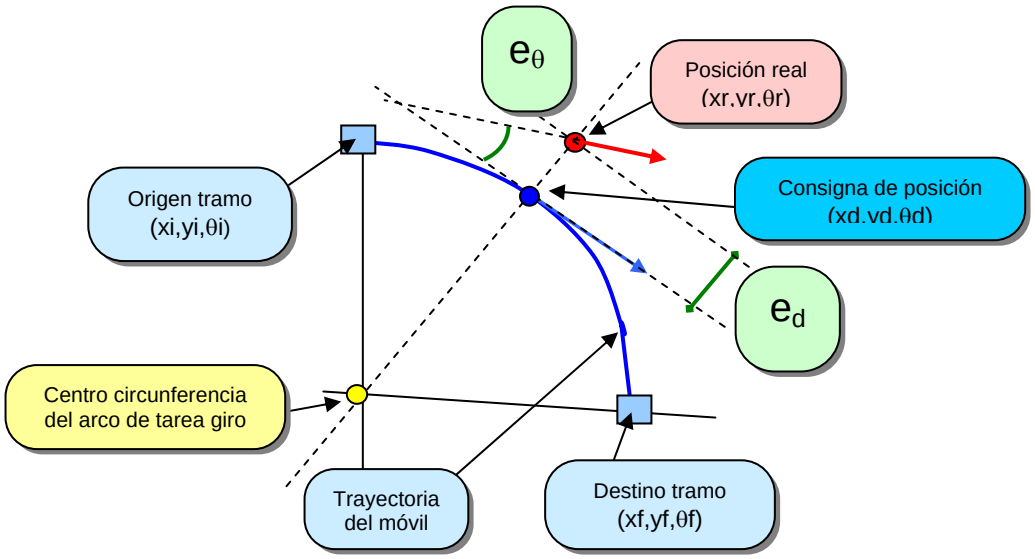


Figura 8.5. Obtención de la consigna de posición en una tarea de giro de ejemplo.

Este método de obtención de consigna es el más empleado en sistemas de control de trayectorias [2], si bien parece bastante atípico el hecho de que la consigna se obtenga a partir de la posición real de la silla de ruedas.

El modo de obtener la consigna depende, por tanto, del tipo de trayectoria que el robot esté siguiendo, que será, por tanto, el primer parámetro a comprobar por el proceso de control en cada período de muestreo. En los apartados siguientes se muestra el algoritmo empleado para las dos trayectorias que, tal y como se veía en el capítulo anterior, puede seguir el robot en este trabajo.

8.4.1. Algoritmo de obtención de la consigna de posición en tareas de avance.

La trayectoria de avance es, matemáticamente hablando, la más sencilla de definir, pues se trata de una línea recta. Por ello la terna de valores que definen la consigna de posición en este caso (*xd*, *yd*, *θd*) queda reducida a dos, pues la consigna de orientación será la propia orientación de la trayectoria a seguir.

El algoritmo que permite obtener estas consignas queda también bastante simplificado, constando básicamente de dos pasos (ver figura 8.5):

1. Obtener *la ecuación normal* a la recta que forma la trayectoria y que pasa por el punto real en que se encuentra el móvil.
2. La consigna de posición se encontrará en la *intersección* de la recta obtenida en el punto anterior y la recta que constituye la trayectoria.

Las ecuaciones 8.10 y 8.11 permiten obtener las coordenadas a partir de los parámetros proporcionados por el generador de trayectorias y de la posición real del móvil.

$$xd = \frac{yi - \tan(\theta i)\cdot xi - yr + \tan(\theta i + \pi/2)\cdot xr}{\tan(\theta i + \pi/2) - \tan(\theta i)} \qquad <8.10>$$

$$yd = \tan(\theta i)\cdot(xd - xi) + yi \qquad <8.11>$$

Para facilitar la lectura se ha empleado la nomenclatura presentada en las figuras anteriores.

8.4.2. Algoritmo de obtención de la consigna de posición en tareas de giro.

La obtención de la consigna en tareas de giro es algo más complicada que la anterior, pues en este caso la trayectoria es un segmento de circunferencia, con lo que será necesario obtener el valor de las tres variables que definen la posición absoluta del móvil (x, y, θ). Sin embargo, gracias a los datos proporcionados por el generador de trayectorias, el cálculo a realizar no es muy extenso.

En la figura 8.5 se muestra de forma gráfica la localización del punto que es consigna de posición para este tipo de trayectorias.

Los pasos a dar para implementar la primera parte del algoritmo son los mismos, pero en este caso hay que obtener también la consigna de orientación:

1. Obtener *la ecuación de la recta* que pase por el punto real en que se encuentre el robot y por el origen de coordenadas de la circunferencia a la que pertenece la trayectoria.
2. Las coordenadas del punto buscado se obtienen de la *intersección* entre esa recta y la circunferencia que constituye la trayectoria.
3. La consigna de orientación se obtiene como la normal a la recta que une el punto real con el centro de la circunferencia.

De nuevo, a partir de esas especificaciones se obtienen fácilmente las expresiones algebraicas (8.12, 8.13 y 8.14) que permiten obtener la consigna de posición en estos casos.

$$yd = yc + \frac{R}{\sqrt{\left(\frac{xr - xc}{yr - yc}\right)^2 + 1}} \qquad <8.12>$$

$$xd = xc + \frac{xr - xc}{yr - yc} \cdot (yd - yc) \qquad <8.13>$$

$$\theta d = \arctan\left(\frac{yc - yr}{xc - xr}\right) + \frac{\pi}{2} \qquad <8.14>$$

En este momento el sistema está en disposición de aplicar el algoritmo de control adecuado.

8.5. Los algoritmos de control.

Observando de nuevo el diagrama general del lazo de control (figura 8.1) se observa que se está ante un sistema SIMO (*Simple Input Multiple Output*), considerando la posición como única entrada del sistema, aunque formada por tres coordenadas.

En esa misma figura se observa que, para afrontar este sistema se ha optado por emplear un controlador distinto para cada variable de salida. Ya en otros trabajos [2] se estudia la posibilidad de aunar las dos variables de salida en un único controlador, pero dicha posibilidad no se implementa, pues dificulta el diseño y el ajuste del mismo sin mejorar ninguna de las salidas.

En la figura 8.2 se describe mediante un flujograma el lazo de realimentación. En este apartado se realiza una descripción detallada del método de diseño y el porqué de cada uno de los elementos que aparecen en la figura anterior.

8.5.1. El control de velocidad de avance (V).

La variable de velocidad de avance (V) es la que permite a la silla avanzar por la trayectoria diseñada por el generador de trayectorias. El controlador de V ha de encargarse de que ésta sea lo más adecuada posible a las características del tramo a recorrer, haciendo que la conducción sea confortable para el usuario.

A partir del objetivo planteado, es necesario elegir la ley de control que permite obtener la actuación más adecuada en cada período de muestreo. En los trabajos estudiados y presentados en el capítulo 2, uno de los controladores más recurridos para la velocidad de avance es el de los controladores borrosos. Estos controladores presentan ciertas ventajas e inconvenientes para la aplicación presentada:

☺ Permiten *incorporar en la ley de control variables que no se pueden modelar* con la planta, como puede ser el efecto de la curvatura de un giro en la velocidad de avance, etc.

☺ Permiten realizar un *ajuste particular de la V para cada situación* concreta del móvil, pues son controladores que, debido a su constitución incorporan distintas leyes de control en un solo proceso.

☹ Sin embargo, la ventaja anteriormente mencionada se vuelve desventaja a la hora *de abordar el ajuste*, que siempre es muy tedioso. Debido a la cantidad de variables que permite incluir en el control, el ajuste de la actuación para cada valor de las mismas a veces se torna en un trabajo complicado, en el que se pierde la visión real de la situación del controlador que se está diseñando.

☹ Por otro lado, la gran cantidad de información con la que ha de trabajar el controlador borroso, y el gran número de pasos que hay que desarrollar con dicha información hace que *el tiempo de cómputo crezca* si no se emplean los procesadores adecuados.

Sopesando las ventajas y los inconvenientes, y con la premisa de proponer una ley de control alternativo, en este trabajo se elige un proceso distinto al control borroso. Se trata de un modelo no lineal que actúa generando un perfil de velocidad acorde con el entorno. En la figura 8.6 se muestra a modo de diagrama de entradas y salidas, el modelo del controlador de velocidad de avance.

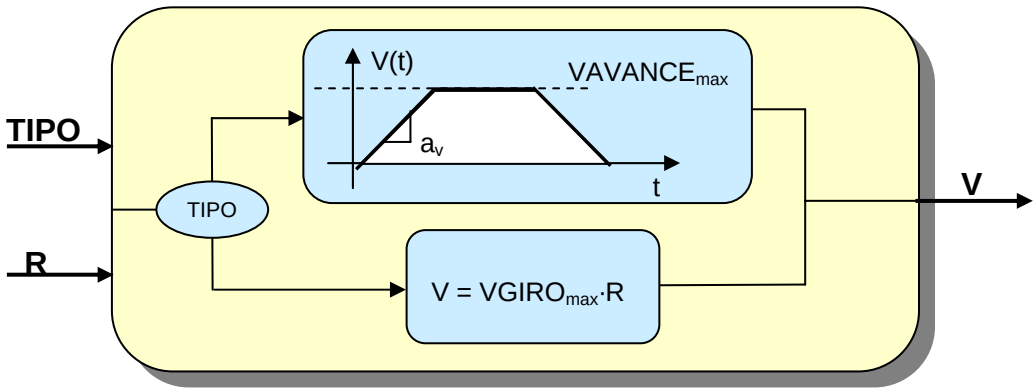


Figura 8.6. Diagrama de bloques del generador de velocidad de avance.

Tal y como se ve en la figura, el modelo toma como entradas los siguientes parámetros que proporciona el generador de trayectorias para describir cada tramo (ver la figura 8.4):

1. *El tipo de tarea (TIPO).* En función del tipo de tarea que se esté trazando, el controlador genera un perfil u otro para la velocidad de avance:
   a) Si *la tarea es de avance* el controlador genera un perfil trapezoidal para la V, en el que el valor máximo y la aceleración son constantes en el algoritmo. De este modo se acelera el avance de la silla en los pasillos largos, y se decelera para afrontar una tarea de giro o al acercarse al destino.

b) Si *la tarea es de giro,* la V será constante y proporcional al radio de curvatura del giro, facilitando el trazado del arco de giro.

2. *El radio de curvatura del giro (R)* de la tarea actual o de la siguiente si la actual es una tarea de avance. Como se ha comentado, en las tareas de giro la V es constante en todo el trayecto y absolutamente proporcional al radio de giro, de modo que si el radio es nulo, la V será también nula, permitiendo a la silla realizar giros sobre sí misma.

Además, tal y como ya se ha comentado, el perfil de V generado depende de otros tres parámetros, que son constantes en todo el movimiento.

a) *La V máxima en tareas de avance ($VAVANCE_{max}$).* Es la máxima velocidad que alcanza la silla en los pasillos largos, pues en este caso la velocidad de giro (Ω) es nula. Este parámetro se fija en función de las características dinámicas de la silla y de la comodidad de conducción.

   Teniendo en cuenta la tensión máxima del puente de excitación de los motores y la ganancia estática de los mismos, se puede obtener que la velocidad de avance absoluta máxima de la silla es de alrededor de los 4m/s, es decir de unos 14,4Kmh, una velocidad bastante elevada para circular por el interior de un edificio. La velocidad máxima se fija, sin embargo, a 0,7m/s o 2,5Kmh, velocidad algo mayor que la de una persona andando a ritmo normal.

b) *La V máxima en tareas de giro ($VGIRO_{max}$).* Es la máxima V que alcanza la silla en una tarea de giro, si bien no permite determinar la rapidez de la silla en giros, pues para ello es necesario tener en cuenta toda la cinemática de la silla, en la que entra también en juego la velocidad de giro (Ω) (ver ecuaciones 8.8 y 8.9).

   Como se ha comentado, esta velocidad es ponderada en función del radio de curvatura del giro, de forma proporcional. El valor ajustado de forma empírica es de 0,2m/s, suficientemente alto como para hacer avanzar al móvil pero no como para tener más peso que la velocidad de giro (Ω) en la tarea de giro.

c) *La aceleración de V (a_V).* Es el parámetro que permite fijar la pendiente de aceleración y deceleración en el perfil trapezoidal generado para V en las tareas de avance. Esta constante ha sido fijada por comodidad a 0,1m/s^2, lo cual significa, p.ej, que avanzando en línea recta y partiendo del reposo, el móvil alcanza la velocidad máxima en 14.3s.

   En cuanto a las tareas de giro este parámetro también se emplea para alcanzar la V proporcional a la curvatura en estas tareas, cuando la velocidad de partida al iniciar una tarea de giro es menor.

Con todos estos parámetros el controlador desarrolla el algoritmo de generación del perfil de V, cuyos resultados se muestran en la figura 8.7. En esta última figura se indica cuándo aparece un cambio de tarea, de modo que se pueden observar fácilmente los efectos del algoritmo en la velocidad de avance.

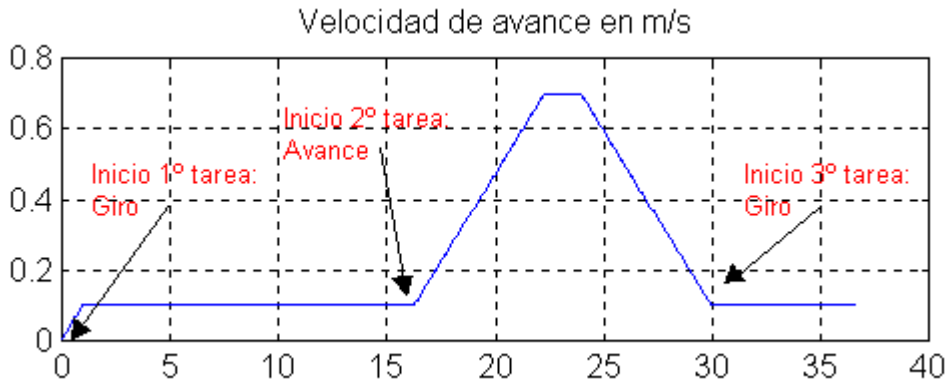


Figura 8.7. Ejemplo de perfil de la velocidad de avance frente al tiempo para distintas tareas.

Visto todo el planteamiento anterior, queda claro que el control de velocidad de avance (V) es totalmente alineal y adaptado a la conducción de sillas de ruedas en interiores, del mismo modo que ocurre en la mayor parte de los trabajos de navegación estudiados. Es, sin duda, el mejor y más empleado método de obtener la actuación de V.

Además, teniendo en cuenta que V controla en gran medida la rapidez de cambio de la velocidad angular de las ruedas, se han creado perfiles suaves para evitar problemas de deslizamiento que provocasen errores en el sistema de posicionamiento absoluto.

8.5.2. El control de velocidad de giro (Ω).

La velocidad de giro (Ω) tiene una función bastante distinta a la planteada para la de avance (V). Esta variable permite al móvil seguir fielmente la forma de la trayectoria a recorrer, minimizando el error de posición que separa la localización real del robot del camino diseñado por el generador de trayectorias en cada período de muestreo.

8.5.2.1. La elección del controlador.

En este caso las opciones empleadas en los trabajos bajo estudio son mucho más variadas: controladores borrosos, técnicas de control óptimo, control clásico de tipo PID, control robusto, etc.

En este trabajo se ha optado por emplear un simple control proporcional, en base a los siguientes planteamientos:

a) Al ser un control de posición la planta es de tipo 1, con lo que no es necesario incluir ningún elemento integrador en el control para asegurar error nulo en régimen permanente.

b) La ventaja de elegir una técnica de control clásico es que exige una menor potencia de cálculo del procesador asociado, pues la complejidad del algoritmo es mucho menor.

c) Por otro lado, la posibilidad de incluir elementos alineales, como los que presenta un controlador borroso, también se puede encontrar en el control clásico, como se pudo observar en la descripción del controlador de V. Para ello se emplean bloques limitadores, filtros de aceleración, o constantes de ponderación en función de ciertos parámetros del entorno.

d) Finalmente las variables internas del sistema que se pueden controlar empleando una estrategia de variables de estado, no son tan interesantes en sistemas como el presentado, pues existen sistemas en el bajo nivel que están más en contacto con estas variables (excitación de los puentes, corriente por el motor, etc.)

A partir de los planteamientos presentados se elige como algoritmo de control un proporcional al error de posición, tal y como se muestra en la figura 8.8 siguiente.

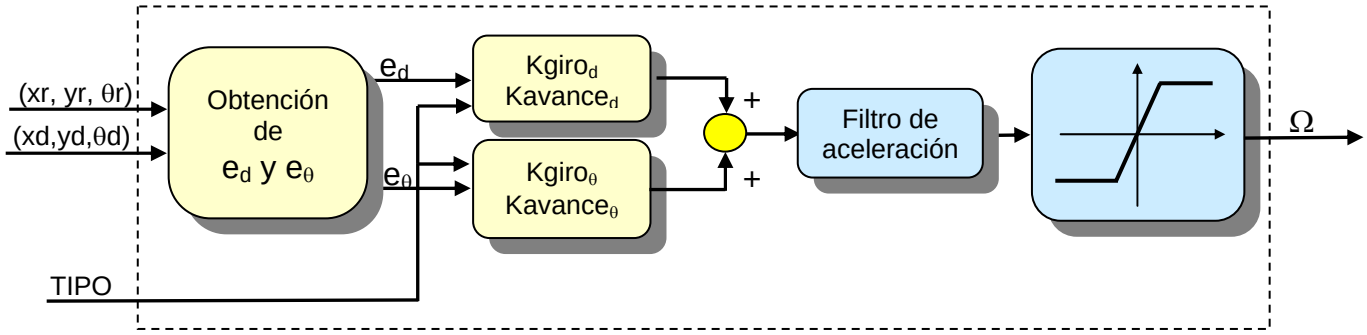


Figura 8.8. Algoritmo de c4ontrol de velocidad de giro.

8.5.2.2. La obtención de las variables de error.

Tal y como se observa en la figura 8.8, las variables de error sobre las que se construye el controlador no se obtienen de la comparación directa de consigna y la posición real, sino que, para simplificar el proceso, se asocian las variables de posición *x* e *y* en una única, que informa del error de desplazamiento del móvil.

En la figura 8.5 se mostraba de forma gráfica el valor de las dos variables de error que se emplean para diseñar la ley de control en este proyecto:

- El error de desplazamiento: e_d.
- El error de orientación: e_θ.

Las ecuaciones 8.15 y 8.16 muestran cómo obtener el valor de las señales de error a partir de la posición real del móvil y la consigna de posición obtenidas previamente.

$$e_d = \sqrt{(xr - xd)^2 + (yr - yd)^2} \qquad <8.15>$$

$$e_\theta = \theta d - \theta r \qquad <8.16>$$

Hay que tener en cuenta que la señal de error de desplazamiento (e_d) pierde el signo, por lo que deberá adoptar el de la señal de error de orientación (e_θ).

La figura 8.10 muestra las señales de error en el seguimiento de una trayectoria como la mostrada en la figura 8.9. Se observa que el error de desplazamiento nunca supera los 10cm, con una media de 2.9 cm, mientras que el error de orientación tiene una media de 0.5 grados. Es necesario observar el hecho de que las oscilaciones del Ω no son apreciables desde el punto de vista del usuario, pues son rápidamente corregidas.

Además en la figura 8.9 se muestra el comportamiento real del sistema completo, donde la línea roja muestra la trayectoria esperada, mientras que la línea azul muestra la respuesta real del sistema completo.

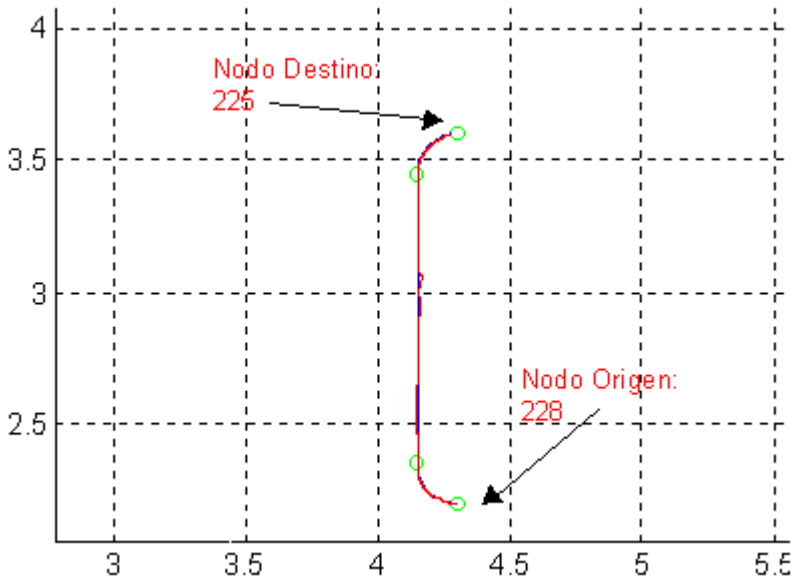


Figura 8.9. Trayectoria de ejemplo con respuesta real del sistema (coordenadas en metros).
Trazo rojo: Respuesta real del sistema.
Trazo azul: Respuesta planificada por el generador de trayectorias.

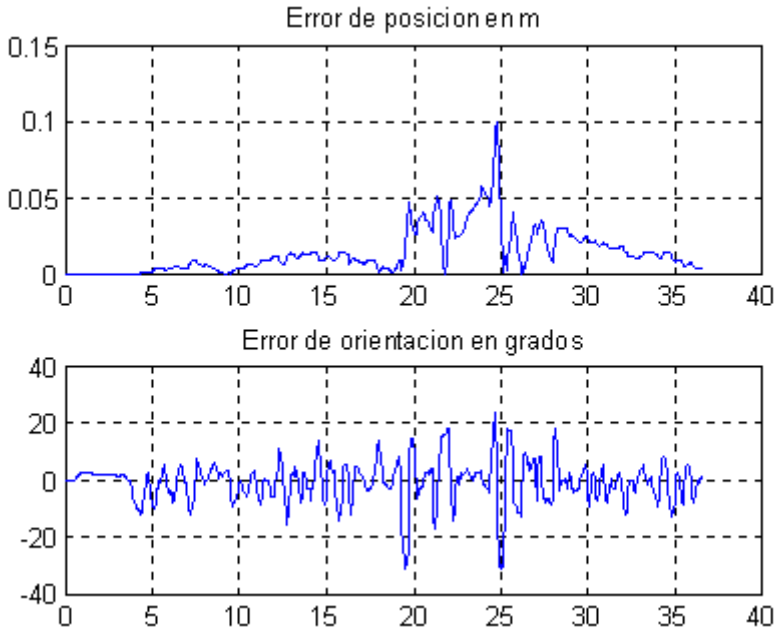


Figura 8.10. Evolución de las señales de error e_V y e_θ frente al tiempo (s) en la trayectoria de ejemplo de la figura 8.9.

Esta técnica presentada no es novedosa en el proyecto, sino que ya se ha empleado en varios de los trabajos estudiados, con lo que está suficientemente fundamentada.

8.5.2.3. El algoritmo de control.

A partir del error obtenido en el apartado anterior se puede aplicar el control proporcional ya comentado mediante la ecuación 8.17.

$$\Omega[n+1] = K_d \cdot e_d[n] + K_\theta \cdot e_\theta[n] \qquad [rad/s] \quad <8.17>$$

El único parámetro de ajuste de este controlador se encuentra en las constantes de proporcionalidad, que harán que la actuación sea mayor o menor en función de los errores. El valor de estas constantes depende, en primer lugar del tipo de tarea que se esté realizando, de modo que el control proporcional varía dando lugar realmente a estas dos expresiones (8.18 y 8.19):

Para tareas de avance:

$$\Omega avance[n+1] = Kavance_d \cdot e_d[n] + Kavance_\theta \cdot e_\theta[n] \qquad [rad/s] \quad <8.18>$$

Para tareas de giro:

$$\Omega giro[n+1] = Kgiro_d \cdot e_d[n] + Kgiro_\theta \cdot e_\theta[n] \qquad [rad/s] \quad <8.19>$$

Las constantes de proporcionalidad se han ajustado de forma empírica, teniendo en cuenta que valores muy altos de estas constantes hacen que el móvil oscile alrededor de la trayectoria deseada. Es por ello que el valor de las mismas ronda siempre los 0.3 grados por cm de error en el caso del error de desplazamiento y los 0.6 grados por grado para el error de orientación.

Un estudio completo del efecto de las constantes se obtendría realizando una simulación del algoritmo de control. Sin embargo, la complejidad del sistema completo ha hecho complicada la construcción de un modelo de simulación, por lo que se pospone esta tarea para posteriores trabajos.

8.5.2.4. Los elementos alineales.

Del mismo modo que ocurre con el controlador de V, el algoritmo completo de control presenta dos elementos alineales a parte del controlador proporcional, tal y como se observa en la figura 8.8 anterior: un filtro de aceleración y un elemento saturador.

1. *El filtro de aceleración,* permite limitar los cambios bruscos de Ω de nuevo, con el fin de evitar deslizamientos que provoquen errores acumulativos en el sistema de deadreckoning.

   En este caso, la limitación está en 1,4rad/s (alrededor de 80grados/s), lo cual permite que se alcance la velocidad máxima de las ruedas (4m/s, tal y como se vio en el apartado 7.4.2) en 18s, cota bastante aceptable para los propósitos establecidos.

2. *El elemento saturador*, limita el valor máximo de giro de la silla del mismo modo que se hace con la velocidad de avance. Bajo las mismas premisas que las expuestas en el punto 7.4.2, la velocidad máxima absoluta de giro de la silla de ruedas sería de 25,13rad/s, lo cual es excesivo a todas luces para la aplicación bajo estudio.

   La velocidad máxima de giro se fija en 4,5rad/s, velocidad que, teniendo en cuenta el valor de aceleración fijado anteriormente, se alcanza en alrededor de 3s, con lo que no se limita la rapidez de respuesta del algoritmo en exceso.

Tal y como se aprecia en la mencionada figura 8.8, el orden de aplicación de los dos módulos ha de ser justamente el que se ha indicado anteriormente: 1º el filtro de aceleración y 2º el elemento saturador. En caso contrario, el efecto del saturador no sería el mismo, pues la limitación de velocidad quedaría por encima de lo deseado.

En la figura 8.11 se observa cuál es la respuesta de Ω una vez implementado todo el sistema de control, para el seguimiento de la trayectoria mostrada en la anterior figura 8.9.

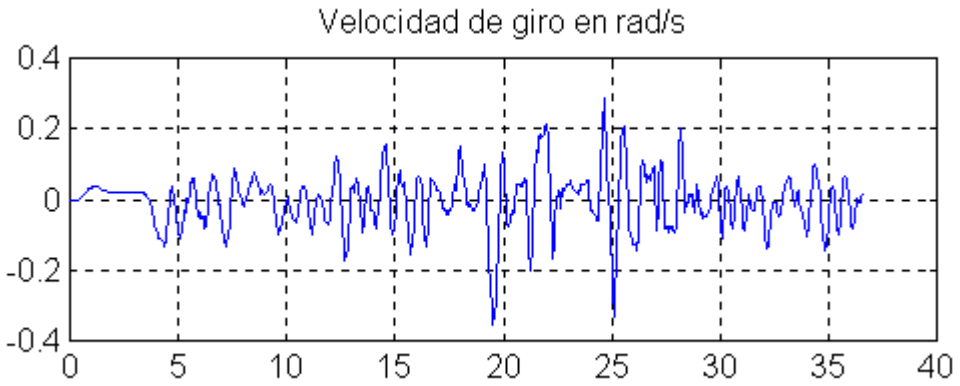


Figura 8.11. Evolución de la señal Ω ante el tiempo (s) para la trayectoria de ejemplo de la figura 8.9.

8.6. El envío de las consignas al bajo nivel.

Una vez que se obtiene la actuación de los dos controladores (V y Ω), es necesario proporcionárselas al sistema de bajo nivel, donde se comportarán como consignas de velocidad.

Sin embargo, antes de enviarlas a la tarjeta de interfaz es necesario aplicar las ecuaciones de cinemática directa (8.20 y 8.21) para obtener las consignas en forma de velocidad angular de las ruedas (ω_D y ω_I) pues es el formato que espera recibir el algoritmo de bajo nivel, tal y como se mostraba ya en la figura 8.1.

$$\omega_D = \frac{1}{R}\cdot\left(V + \Omega\frac{D}{2}\right)\cdot 1000 \qquad [mrad/s] \qquad <8.20>$$

$$\omega_I = \frac{1}{R}\cdot\left(V - \Omega\frac{D}{2}\right)\cdot 1000 \qquad [mrad/s] \qquad <8.21>$$

donde R es el radio de la rueda, D es la distancia entre ruedas y las velocidades angulares se expresan en mrad/s para obtener una mayor resolución al tratarlas como enteros en el sistema de bajo nivel.

Con esta información, el sistema de control está listo para acceder a la zona de interfaz adecuada, donde se volcarán los datos del modo que se muestra en la tabla de la figura 8.12. Como se muestra en la tabla, en el intercambio se incluye una bandera que se activa cuando existen nuevas consignas para el control de bajo nivel. Este flag

permite optimizar el trasiego de información por la red, tal y como se verá en el próximo capítulo 9.

Parámetro	Identificación
ω_D	Consigna de velocidad angular para la rueda derecha (mrad/s)
ω_I	Consigna de velocidad angular para la rueda izquierda (mrad/s)
FLAG	Flag de indicación *"nueva consigna"*

Figura 8.12. Tabla con descripción funcional de los parámetros pasados al bajo nivel tras ejecutar el algoritmo de control.

9. La Plataforma Hardware.

Planteado todo el sistema de navegación, el paso siguiente es el de llevarlo a la plataforma hardware sobre la que se va a implementar.

En este capítulo se describe la arquitectura del sistema de navegación completo, mostrando la ubicación de los algoritmos de alto nivel diseñados, las características más relevantes del sistema de bajo nivel sobre el que se desarrollan y el modulo sobre el que se desarrolla el protocolo de comunicación existente entre ambas partes.

A lo largo de todo el capítulo se manejan términos como autonomía eléctrica, facilidad de transporte, flexibilidad en la adición de nuevos módulos, etc., pues un punto de vista que no se pierde desde el principio del desarrollo es que la plataforma final ha de ser compacta para facilitar su uso en el entorno de un discapacitado [8].

En el proyecto se ha intentado diseñar una plataforma que cumpla todas las características mencionadas, a la vez que incluya la capacidad de cómputo y el tiempo de respuesta deseados. No hay que olvidar que se desea completar una plataforma previa [5] basada en una arquitectura abierta, con la intención de que en posteriores trabajos se mejore el sistema completo, tal y como ya se ha mencionado a lo largo de la memoria.

9.1. Descripción general de la plataforma hardware del sistema.

En la figura 9.1 se muestra un diagrama general de la plataforma hardware del sistema completo, mientras que en la figura 9.2 aparece una fotografía real del sistema de navegación en movimiento.

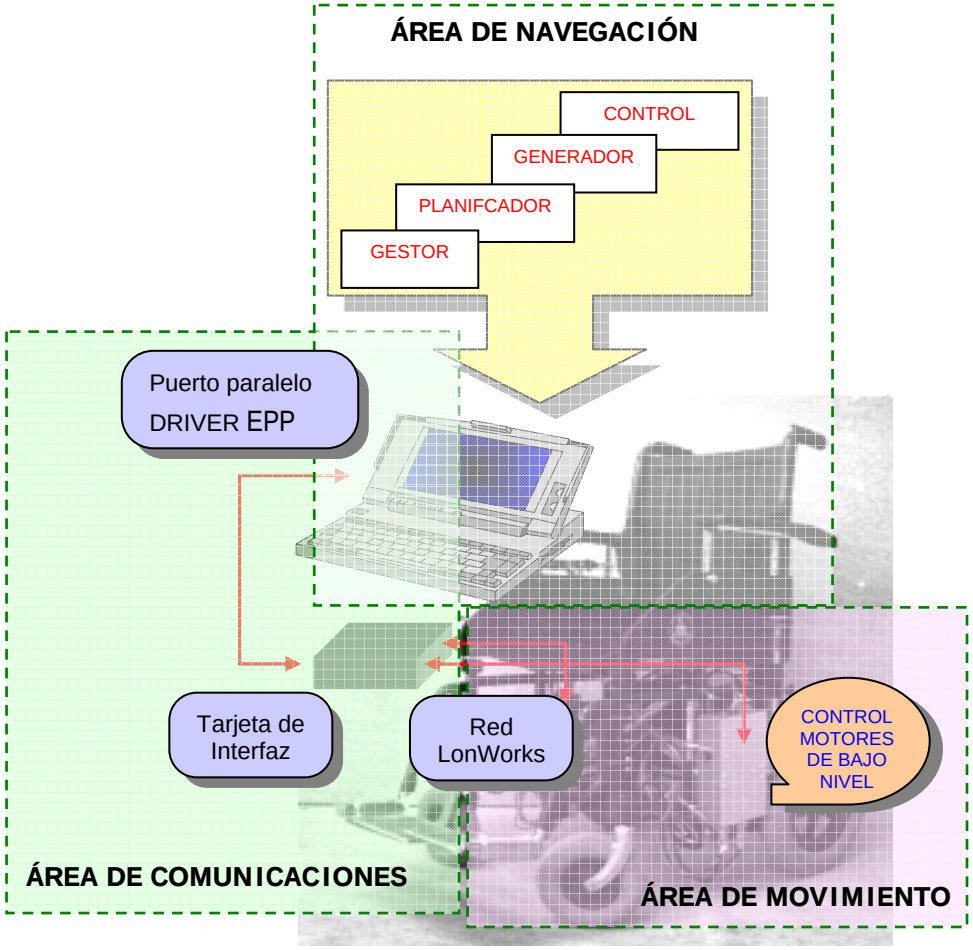


Figura 9.1. Diagrama de bloques del sistema completo.

En la figura 9.1 se aprecian perfectamente las tres partes que se pueden distinguir en el sistema construido:

1. *Área de navegación (alto nivel),* en la que se incluyen todos los procesos de navegación descritos hasta ahora. Físicamente se centraliza en un PC portátil que va sobre la silla de ruedas y que sirve tanto de cerebro de navegación como de interfaz de usuario. Las consignas de velocidad obtenidas del sistema de navegación son proporcionadas al bajo nivel a través del puerto paralelo del PC.

2. *Área de movimiento (bajo nivel),* formada por los dos motores de la silla, junto con el resto de hardware de excitación y software de control de bajo nivel. Físicamente

se localiza en diversas tarjetas desarrolladas en un proyecto anterior [5], y cuyo cerebro de cálculo y comunicación se encuentra en los procesadores de Echelon, los Neuron Chip.

3. *Área de comunicaciones (interfaz),* que permite unir los dos elementos anteriores, que trabajan de forma asíncrona, con una gran velocidad de trasferencia y con control de coherencia de datos.

Figura 9.2. Fotografía del sistema de navegación en movimiento.

Este último bloque es el que permite que el lazo de control, iniciado por el sistema de navegación se cierre. En la tabla de la figura 9.3 se muestra un resumen de la información que es intercambiada periódicamente entre el navegador y el control de bajo nivel, así como del sentido en que fluye dicha información.

Parámetro	Identificación	Sentido
T_D	Acumulación de tiempo de la rueda derecha	PC->Bajo Nivel
T_I	Acumulación de tiempo de la rueda izquierda	PC->Bajo Nivel
P_D	Acumulación de pulsos de la rueda derecha	PC->Bajo Nivel
P_D	Acumulación de pulsos de la rueda derecha	PC->Bajo Nivel
ω_D	Consigna de velocidad angular para la rueda derecha (mrad/s)	Bajo Nivel->PC

ω_I	Consigna de velocidad angular para la rueda izquierda (mrad/s)	Bajo Nivel->PC
FLAG	Flag de indicación *"nueva consigna"*	Bajo Nivel->PC

Figura 9.3. Tabla resumen de la información intercambiada entre el sistema de navegación y el bajo nivel.

Es importante recalcar que el intercambio de información se realiza de forma asíncrona, pues los procesadores que generan la información trabajan con periodos de repetición distintos:

- *El módulo de navegación* genera una consigna de velocidad angular para cada rueda cada 50ms (período de muestreo del sistema de alto nivel, ver capítulo 8).
- *El módulo de control de bajo nivel*, sin embargo, genera una lectura de encoders cada 14ms (pues ese es el período de muestreo a este nivel), que el sistema de navegación leerá para obtener la velocidad real de las ruedas, y completar así el proceso de deadreckoning.

Para sincronizar estas dos tareas se emplea una memoria Dual Port, que será gestionada mediante semáforos desde ambas partes en cada acceso a la información:
- Por una lado el PC dispone de un driver que le permite acceder a la memoria a través del puerto paralelo (modo EPP);
- por otro lado, una tarjeta con un Neuron Chip se añade a la arquitectura distribuida del bajo nivel para permitir el paso de la información de la memoria a la red LonWorks, en la que se encuentran las tarjetas de control de motores.

Debido a este planteamiento, el camino que ha de recorrer la información para moverse entre el procesador de alto nivel (PC) y los de bajo nivel (Neuron Chips) incluye dos canales distintos:
1. *El cable paralelo* a través del que se accede desde el PC a la memoria de intercambio, encontrándose la información en formato byte.
2. *El cable de par trenzado* sobre el que se soporta la comunicación LonWorks entre los módulos motores y el Neuron Chip que accede a la memoria Dual Port. La información se encuentra en este caso en el formato típico del protocolo LonWorks, la variable de red.

Es el Neuron Chip del interfaz el encargado de realizar la traducción entre ambos formatos, tal y como se verá más adelante.

En la figura 9.4 se presenta un diagrama que muestra de forma gráfica el camino de intercambio de datos.

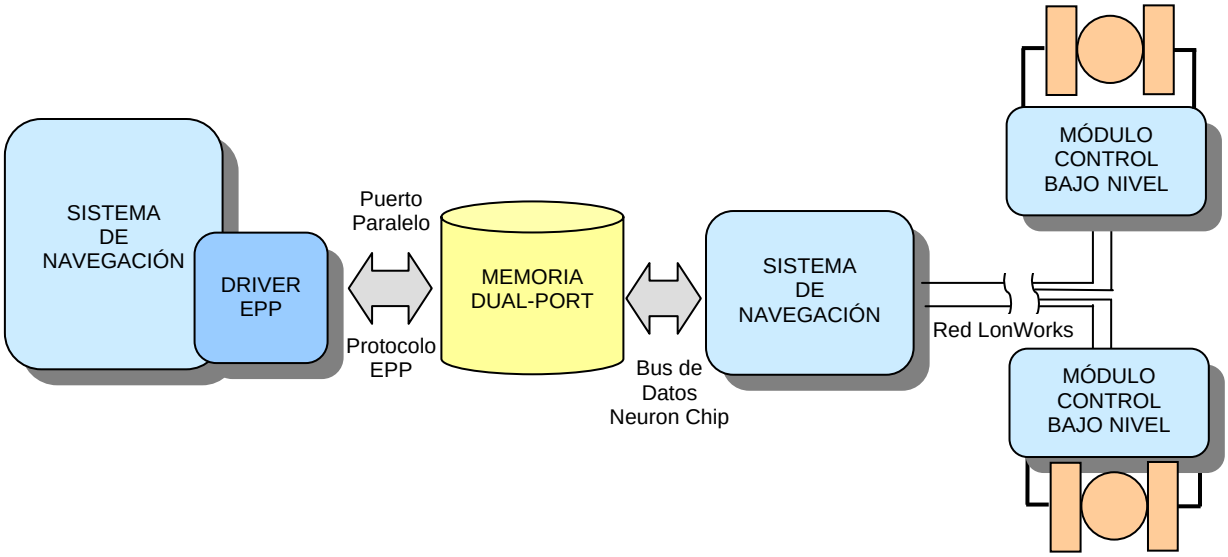


Figura 9.4. Diagrama de intercambio de datos a nivel hardware entre el alto y el bajo nivel del sistema de navegación.

A lo largo del capítulo se describe detalladamente todo el proceso de intercambio de datos implementado en el módulo de comunicaciones, así como las características más relevantes del área de bajo nivel, completando de este modo la descripción global del sistema de navegación autónoma.

9.2. La silla de ruedas y el control de bajo nivel.

Como ya se ha comentado previamente, el sistema de bajo nivel ya fue diseñado anteriormente en otro proyecto [5], por lo que en este apartado se hace referencia a los aspectos más interesantes que incluye o a los elementos añadidos en el desarrollo del proyecto.

En el proyecto antes mencionando, el sistema diseñado incluye varios módulos (módulo de conducción por soplo y por joystick, display informativo, bumpers de efecto reactivo, etc.) que no se han incorporado en este caso, por no ser interesantes para el proyecto desarrollado. El diagrama de la figura 9.5 muestra el sistema de bajo nivel sobre el que se ha construido el navegador.

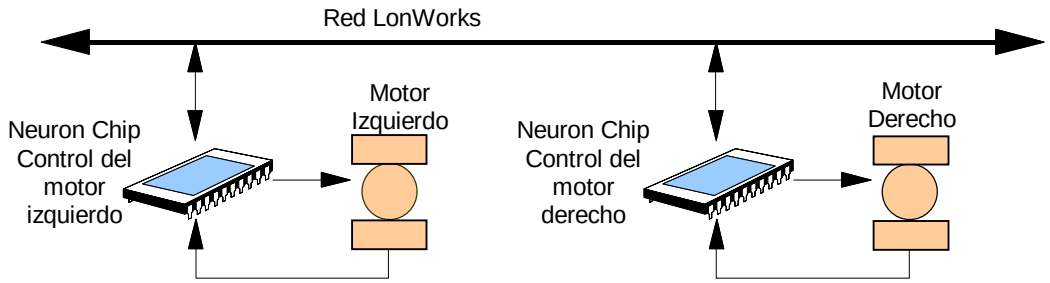


Figura 9.5. Diagrama de bloques de la plataforma de bajo nivel empleada.

Como se observa en dicha figura, el sistema se reduce a las dos tarjetas de excitación de los motores de continua, que junto a los correspondientes módulos de control aseguran que las consignas de velocidad angular de las ruedas se sigan fielmente en los motores.

Además, en la figura 9.5 aparece también la red LonWorks, a través de la cual se comunican los módulos de control con la tarjeta de interfaz, mediante el Neuron Chip de interfaz.

9.2.1. La red LonWorks.

En la última etapa de investigación del departamento sobre la silla de ruedas, todos los proyectos se han realizado en base a la arquitectura LonWorks, que permite la construcción de sistemas inteligentes de control distribuido de un modo sencillo (pues se programan a alto nivel y todo el soporte de gestión de la red queda transparente al desarrollado) y aporta la fiabilidad y velocidad que un sistema de control de este estilo requiere en las comunicaciones.

El núcleo de toda red LonWorks es el Neuron Chip, procesador de 8 bits diseñado para aplicaciones de control, que posee tres CPUs (dos de ellas dedicadas a tareas de red), un sistema operativo en tiempo real y una biblioteca de funciones para sus pines I/O, que le permite adaptarse a multitud de aplicaciones de actuación y control de procesos.

La plataforma de partida presenta una arquitectura de este tipo, y los procesadores que se encargan del control de bajo nivel son Neuron Chips, al igual que el núcleo de la tarjeta de interfaz. Como se verá más adelante, esta tarjeta supone una nueva posibilidad de enlace entre el PC (a través de su puerto paralelo) y la red LonWorks, mediante un sistema que mantiene la fiabilidad del conjunto y no supone un cuello de botella en las comunicaciones.

Debido al soporte de comunicaciones que incluye el Neuron Chip, el intercambio de datos se realiza simplemente mediante la lectura y la actualización de variables. En el diagrama de la figura 9.6 se muestra el diagrama de conexiones existentes entre los dos módulos motores y el nodo de interfaz.

Es importante tener en cuenta la periodicidad con la que transitan los mensajes por la red. Hay 4 variables que circulan por la red cada 14ms, son las variables que permitirán al navegador obtener la posición real de la silla, que son actualizadas, y por tanto enviadas por la red cada período de muestreo en las tarjetas de control de bajo nivel. El tránsito de estos mensajes es asíncrono, pues así lo son las tareas, con lo que se asegura que la carga en la red no sea muy elevada en cada periodo de ejecución.

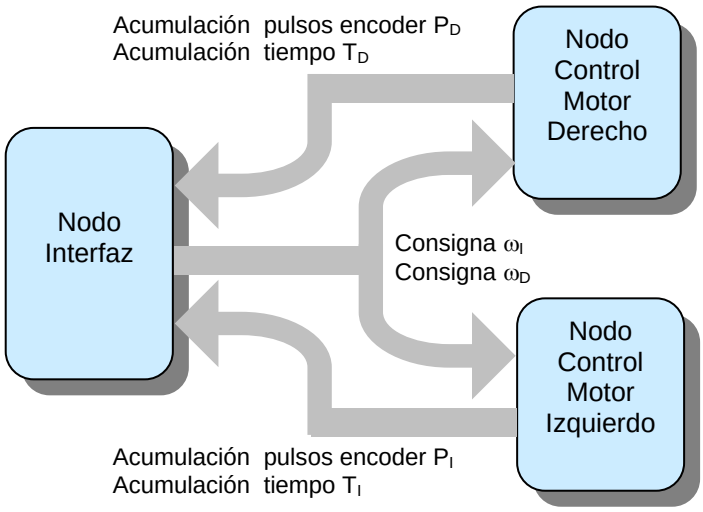


Figura 9.6. Diagrama de conexiones de la red LonWorks.

Por otro lado, en el diagrama de la figura 9.6 aparecen también variables que llegan a los módulos de bajo nivel procedentes del sistema de navegación que se encuentra en el PC. Son las consignas de velocidad angular (ω_D y ω_I), que llegan aproximadamente con la misma periodicidad que son generadas (cada 50ms), tal y como se verá más adelante.

Además aparece también hacia los nodos motores una variable proceden del PC que permite parar los motores en caso necesario, con lo que el flujo de información en la red queda completo.

9.2.2. Los módulos de control de motores.

Al margen del control de posición llevado a cabo por el proceso de control del PC, las tarjetas Neuron Chip colocadas en los motores llevan a cabo un control de velocidad sobre los mismos, asegurando de esta forma que se sigan las consignas de velocidad proporcionadas por los algoritmos de navegación.

El control realizado a bajo nivel es un control clásico PI, basado en la realimentación que proporcionan los encoders. Además del controlador, el software existente incluye:

1. El algoritmo de gestión de la señal del freno eléctrico, que va acoplado a los motores.
2. La generación de la señal modulada en anchura para el puente de transistores que excitan al motor.
3. El conteo de pulsos de las señales en cuadratura procedentes del encoder.
4. El tratamiento de las señales de sobrecorriente que proporciona el driver del puente de excitación.

Todos estos elementos fueron diseñados, depurados y puestos en marcha en el proyecto ya mencionado, y han sido tomados tal cual estaban para este proyecto. El único cambio que se ha hecho ha sido el de aumentar el período de muestreo del sistema de bajo nivel a 14ms, pues de forma experimental se comprobó que el jitter introducido por el temporizador que generaba el proceso de control era demasiado grande para el período que había sido fijado anteriormente (10ms).

Aumentar ligeramente este tiempo hace que el jitter disminuya considerablemente [3], y, por otra parte, no influye de un modo apreciable en el comportamiento del lazo de control de bajo nivel.

9.3. La tarjeta de interfaz.

Uno de los grandes retos del proyecto ha consistido en buscar un método adecuado para comunicar el PC, donde reside el sistema de navegación, con las tarjetas que llevan el control de bajo nivel de los motores.

Varias veces se había planteado este problema anteriormente [1], y hasta el momento se había empleado una tarjeta de interfaz propuesta por los propios diseñadores del protocolo LonWorks (la PCLTA), que no proporcionaba buenos resultados, pues con ella se conseguían tiempos de respuesta de alrededor de 100ms, lo cual ralentizaba enormemente cualquier algoritmo que se hubiese diseñado en el PC.

Posteriormente se planteó la posibilidad de diseñar una tarjeta que, empleado el puerto paralelo del PC y un elemento propio de compartición de información de sistemas multiprocesador, permitiese aumentar la velocidad de trasiego de información. Con esta idea se realizó un proyecto [18] que consistió en el diseño de una tarjeta con una Dual Port a la que acceden, por un lado, un Neuron Chip conectado a la red LonWorks y, por el otro, el PC a través de su puerto paralelo, empleando para el acceso el modo EPP (Enhaced Parallel Port) del mismo.

La tarjeta diseñada se muestra en la figura 9.7, donde se aprecian los componentes básicos que posee, esquematizados en la figura 9.8:

1. Una PAL que permite implementar la lógica de acceso a la memoria compartida tanto desde el puerto paralelo del PC como desde los buses del Neuron Chip.

2. Una memoria Dual Port de 16Kbytes que incluye tres métodos de control de acceso para evitar problemas de coherencia de datos. Se trata de la memoria CY7C006 de Cypress.

3. Un conjunto de buffers y latches que permiten realizar el acoplo de buses entre los procesadores.

En objetivo de este documento no es el de explicar de un modo exhaustivo las bases de funcionamiento del método de comunicación propuesto, sino el modo de empleo de la tarjeta diseñada, así como la aplicación diseñada en este trabajo para acceder a los datos desde ambas partes.

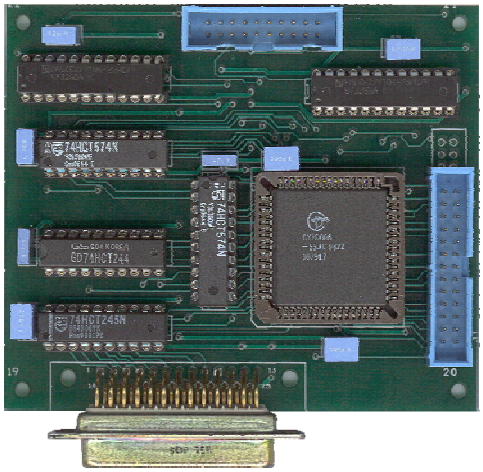


Figura 9.7. Tarjeta de interfaz entre la red LonWorks y el puerto paralelo del PC en modo EPP, basada en una Dual Port.

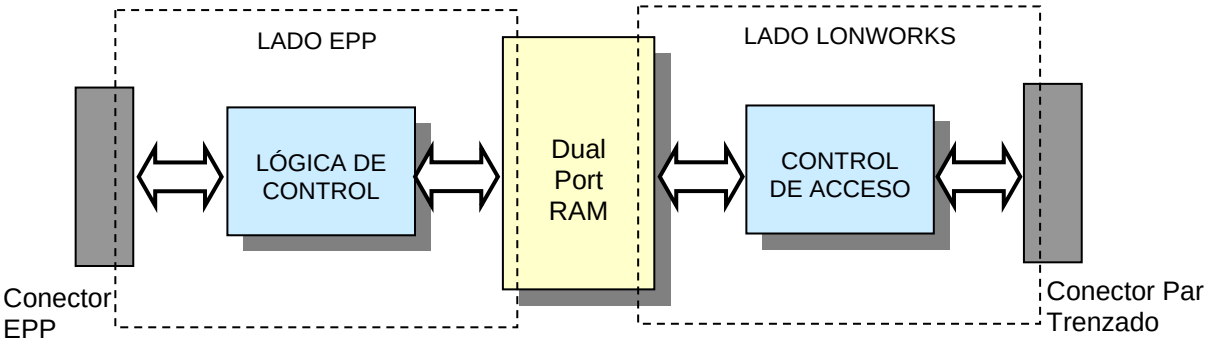


Figura 9.8. Diagrama de bloques del hardware implementado en la tarjeta de comunicación.

9.3.1. Conceptos generales sobre el hardware de la tarjeta de interfaz.

El hecho de emplear una Dual Port en la tarjeta de interfaz facilita la comunicación entre procesos asíncronos como los aquí mencionados, pues permite intercambiar información independientemente de la velocidad de proceso de cada uno, y evita los problemas de bloqueo que daría una memoria estándar empleada para compartir datos

al intentar acceder los dos procesadores a la vez. De hecho es uno de los recursos más empleados para comunicar sistemas multiprocesador.

Además, las memorias Dual Port incluyen diversos métodos que aseguran la coherencia de datos en caso de se desee acceder a la vez a la misma celda de memoria desde los dos puertos, realizando uno de ellos una operación de escritura. Los métodos más característicos son: las líneas de interrupción, las señales de busy y los semáforos hardware.

Para agilizar el acceso a la memoria y facilitar el hardware de decodificación, a la hora de diseñar la tarjeta se seleccionó como método de control de acceso a la memoria el de semáforos hardware. La Dual Port bajo estudio posee una zona específicamente diseñada para soportar la gestión de los semáforos.

En la figura 9.9 se muestra la localización de la memoria Dual Port desde ambas partes de la interfaz. Se observa que existe un offset en la posición que ocupan los 16Kbytes de comunicación desde el punto de vista del Neuron Chip. Esto se debe a que la memoria se ha mapeado en el área de entrada/salida del microprocesador.

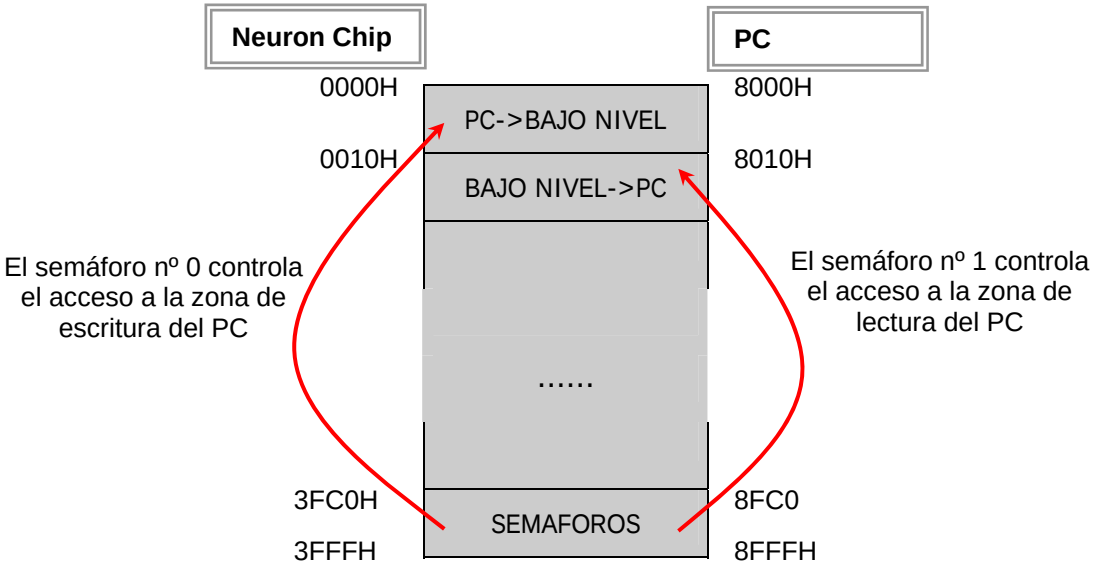


Figura 9.9. Distribución de la memoria Dual Port y localización desde Neuron Chip, y desde el punto de vista del PC.

En la misma figura se observa además la localización de los semáforos, en los últimos 64 bytes de la misma, dejando inutilizada esa parte de la Dual Port para intercambio de datos.

Para poder acceder a una zona de memoria empleando este sistema de protección, es necesario realizar previamente una comprobación de que el semáforo asociado al área deseada está desactivado, para ello el proceso a seguir es el siguiente:

a) Se escribe un '0' en el semáforo correspondiente.
b) Se vuelve a leer dicho semáforo.
c) Si el valor devuelto es '0', el sistema posee el semáforo, por lo que podrá acceder a la zona seleccionada.
d) Si por el contrario el valor devuelto es un '1' el semáforo está indicando que dicha zona está ocupada, por lo que el procesador tendrá que volver a acceder en otro momento.

Hay que tener en cuenta que los semáforos no son más que un recurso que la memoria pone a disposición del usuario, pero que no es paso obligado para acceder a las celdas, por lo que la elección de la zona que gestiona cada semáforo es también decisión del usuario.

Es por ello que para la aplicación de navegación se estimó oportuno emplear dos zonas de la memoria, una para cada sentido del tránsito de datos, a las que se les asignó sendos semáforos. En la figura 9.9 se muestra también la localización de las dos zonas de memoria empleadas en la comunicación, así como el semáforo asociado a cada una de ellas.

Con todo ello quedan sentadas las bases para realizar los accesos a la memoria de doble puerto, que serán realizados como se indica en los siguientes apartados.

9.3.2. El acceso desde el PC. El modo EPP y el driver diseñado.

El modo EPP (parte del estándar IEEE-1284) fue seleccionado por soportar un modo de comunicación bidireccional a una velocidad que varía entre los 500Kbps y los 2Mbps. Si bien no se va a entrar en detalle acerca del funcionamiento del puerto paralelo en este modo de configuración, cabe decir que estas características se deben a que incorpora un driver que permite realizar el protocolo por hardware.

Todas el protocolo de acceso a la memoria de compartición de datos mediante las líneas del puerto paralelo está implementado en la PAL que se encuentra en la tarjeta de interfaz, de modo para el programador el acceso a la memoria se reduce a un acceso normal al puerto con las funciones *inport y outport.*

Además, para facilitar el manejo de la información almacenada en la Dual Port, junto a la tarjeta de interfaz y en el mismo proyecto se diseñó una biblioteca de funciones, que permite realizar transferencias a la memoria en distintos formatos: 1, 2 y 4 bytes, en coma fija y coma flotante, y como variables individuales o en arrays unidimensionales.

Todas estas funciones emplean una sintaxis sencilla y común a todos los formatos:

```
void getIntArray(int address, float data[], int length)
```

La dirección puede estar comprendida en un rango de 16Kbytes, exceptuando las 64 últimas posiciones, donde se encuentran localizados los semáforos (ver figura 9.9).

Además el mismo software incluye funciones de testeo y borrado de la memoria, y las necesarias para capturar y liberar los semáforos asociados a las distintas zonas de la Dual Port. Las funciones asociadas con los semáforos sólo requieren para su funcionamiento el número del semáforo sobre el que se desea actuar.

Teniendo en cuenta todo lo planteado, y sabiendo que cada zona está supervisada por un semáforo, como se mostraba en la figura 9.9, se puede abordar el proceso que tiene lugar en el PC para acceder a los datos. Desde las aplicaciones de navegación sólo existen dos accesos al bajo nivel:

1. Para recoger los datos del tiempo y pulsos de encoder que permitan conocer mediante, el proceso de deadreckoning, la posición real del móvil en cada instante.

2. Para enviar las nuevas consignas de velocidad angular obtenidas tras el proceso de control.

En los diagramas de las figuras 9.10 y 9.11 se muestra el procedimiento seguido en ambos casos, empleado para ello las funciones del driver ya comentadas.
Hay un aspecto en el protocolo de escritura de consignas que merece la pena ser comentado, y es el hecho de que actualizar el valor de las consignas en la Dual Port conlleva la activación de un flag que le indica al Neuron Chip que debe enviar esas nuevas consignas por la red LonWorks. De este modo se disminuye el tráfico por la red, pues solamente cuando exista una nueva actualización de las mismas serán transferidas a los módulos motores.

Además es necesario comentar también que, durante el proceso de inicialización del software de navegación, se incluyen varias funciones de configuración de la parte de comunicaciones:

a) La función *initPort*, que permite realizar la inicialización de los registros del puerto y de los semáforos de la memoria, que quedan liberados

b) La función *clearMem*, que realiza un borrado completo de la memoria, comprobando a la vez la integridad de la misma.

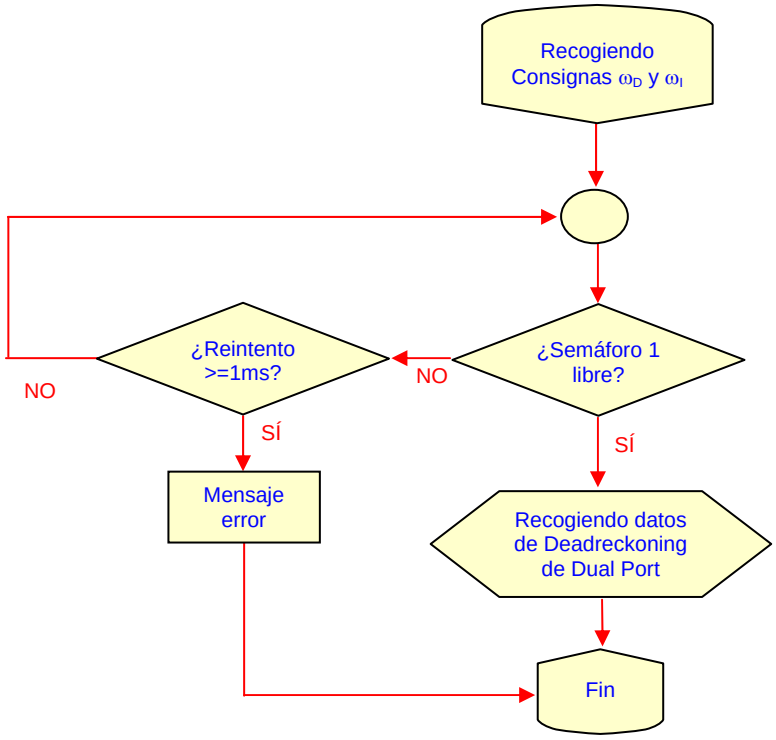


Figura 9.10. Diagrama de bloques del proceso de recogida de datos de la Dual Port para el deadreckoning

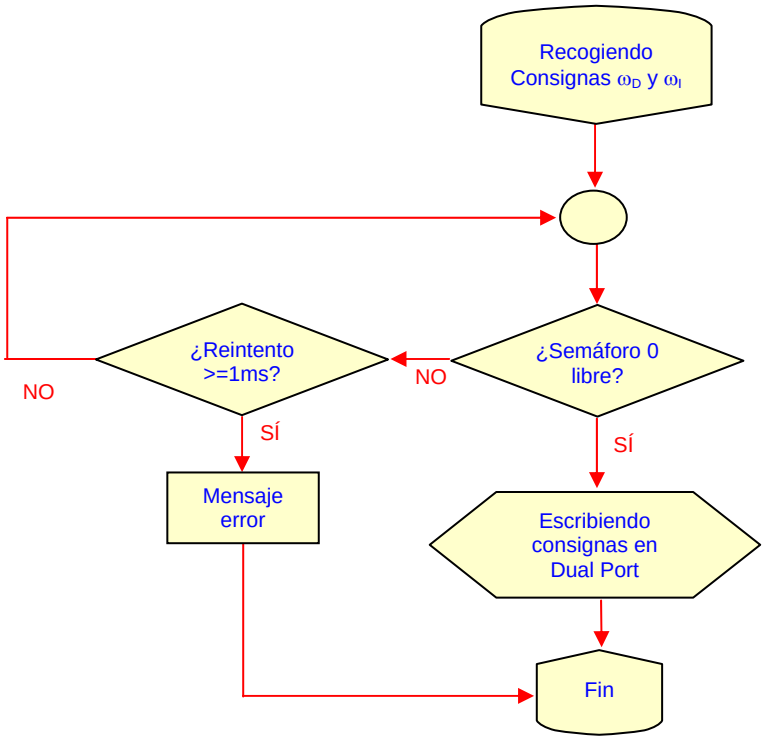


Figura 9.11. Diagrama de bloques del proceso de envío de consignas a la Dual Port.

Como comentario final, cabe decir que las pruebas realizadas muestran que la velocidad de envío de variables alcanza los 700Kbps, lo que supone 2.6μs por variable de tipo entero transferida.

9.3.3. El acceso desde el Neuron Chip.

Desde el Neuron Chip, que hace de puente entre la Dual Port y la red LonWorks, el acceso se realiza mediante un proceso de mapeado en memoria. Tal y como ya se comentaba, se emplea para ello una zona del mapa de memoria del Neuron Chip habilitada para accesos I/O.

El hecho de que la memoria de intercambio esté ubicada en el mapa de memoria del microprocesador, unido a que la lógica de decodificación se implementa por hardware en la PAL, hacen muy sencillo el acceso a la misma, no siendo necesario ningún tipo de driver para ello. La única precaución que hay que tener para acceder a esta memoria es la forma de definir las variables, pues es necesario acceder a esta zona a través de un puntero constante a una estructura, tal y como se muestra a continuación.

```
const Semaphores SemNo = 0xBFC0
```

Tanto los semáforos (tal y como se muestra en el ejemplo) como los propios datos son definidos de esta forma, de modo que cada elemento de la estructura direccionada por el puntero es una variable.

Tal y como ya se comentaba en el apartado anterior, para disminuir la carga de variables por la red LonWorks, existe además de las variables a intercambiar, un flag escrito por el software del PC que permite saber cuando existe una nueva consigna a enviar por la red.

La aplicación completa que lleva a cabo el Neuron Chip se basa en dos eventos que provocan el mismo proceso, cuyo diagrama de bloques se muestra en la figura 9.12. Los procesos vienen desencadenados por la llegada de una variable de los módulos motores informando de la posición de los mismos. La idea consiste en que cada vez que llegue una de estas variable se consulte el flag que informa de que existe una nueva consigna de velocidad en la memoria, y sea entonces cuándo se recoja esta consigna.

Teniendo en cuenta el período de muestreo de los módulos motores, es seguro que al menos una vez cada 50ms (período de muestreo del proceso de alto nivel) se sondea la bandera de *"consigna nueva",* con lo que no existe pérdida de información y se evita tener que estar continuamente sondeando la memoria.

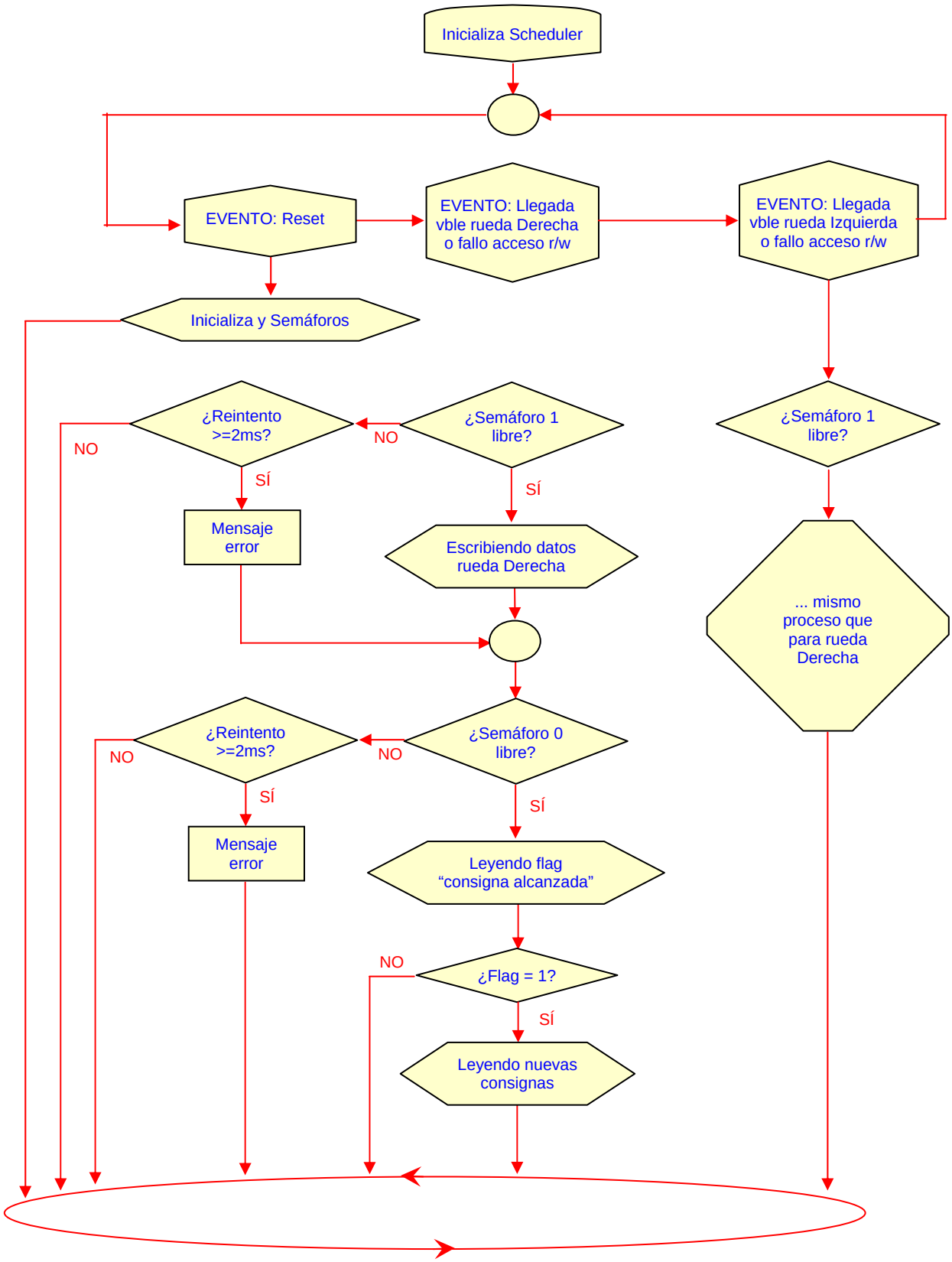


Figura 9.12. Diagrama de bloques del proceso llevado a cabo en el Neuron Chip de comunicaciones.

Otro aspecto interesante del diagrama de la figura 9.12 anterior es que al no existir driver en el bajo nivel, los datos se recogen de la memoria directamente, sin preprocesar, por lo que para obtener las consignas será necesario reconstruir el dato de dos bytes a partir de lo recogido de la memoria.

Además de los eventos comentados, en el programa del Neuron Chip también se incluye la inicialización de los semáforos, liberándolos todos al iniciar el proceso, tal y como hacía también el gestor de procesos con el otro lado de la memoria.

10. Resultados, Conclusiones y Trabajos Futuros.

Desarrollado y expuesto todo el sistema sólo queda mostrar los resultados y conclusiones que se extraen del mismo. En este capítulo se muestran mediante gráficas obtenidas de ejecuciones en tiempo real, el resultado de aplicar el sistema de navegación montado sobre la plataforma presentada.

A lo largo de todo el trabajo, además, han sido continuamente mencionadas posibles ampliaciones del mismo que se recopilan también en este capítulo.

Finalmente, se vuelve en este capítulo a reincidir sobre el hecho de que se trata de un diseño para una aplicación específica, no queriendo ser en ningún momento un estudio profundo de investigación acerca de la navegación autónoma.

10.1. Resultados.

En este apartado se va a mostrar el resultado de ejecutar el sistema de navegación sobre la plataforma autónoma en distintas ubicaciones del edificio bajo estudio. Para mostrar los resultados se ha ido recogiendo información de posición de la salida de los distintos procesos de navegación, y se han representado en gráficas de Matlab los resultados más interesantes.

Las pruebas han sido desarrolladas con un usuario montado sobre la silla, que reconoció un movimiento cómodo, si bien algo lento en algunas ocasiones. Esto es debido sin duda a que durante los giros, la velocidad de avance es bastante baja, pues se pretenden evitar errores de posición del deadreckoning y posibles colisiones en las salidas o entradas a través de puertas. Con un sistema de posicionamiento añadido para las operaciones de giro, el movimiento mejoraría de forma apreciable.

En las figuras que se muestran a continuación se han presentado distintas trayectorias a lo largo del edificio, intentando en todo momento abarcar el mayor número de posibilidades del navegador presentadas en la memoria. Sin embargo, las opciones son tan amplias que sería imposible, a la vez que inapreciable presentar en esta memoria todos los distintos recorridos posibles.

En las figuras se muestra tanto el recorrido esperado de la silla de ruedas, como el real, por lo que fácilmente se obtiene la gráfica de error de posición. Hay que remarcar que el error obtenido es del todo inapreciable para el usuario, y repetir que el objetivo del trabajo no es el de seguir una trayectoria lo más fielmente posible, sino el de generar un camino cómodo para el movimiento de una silla de ruedas.

10.1.1. Recorridos cortos.

En la figura 10.1 se muestra el recorrido realizado por la silla para ir de un laboratorio a otro en el interior de un pasillo de la planta 1. En esta figura se aprecian los distintos movimientos, tanto de giro como de avance.

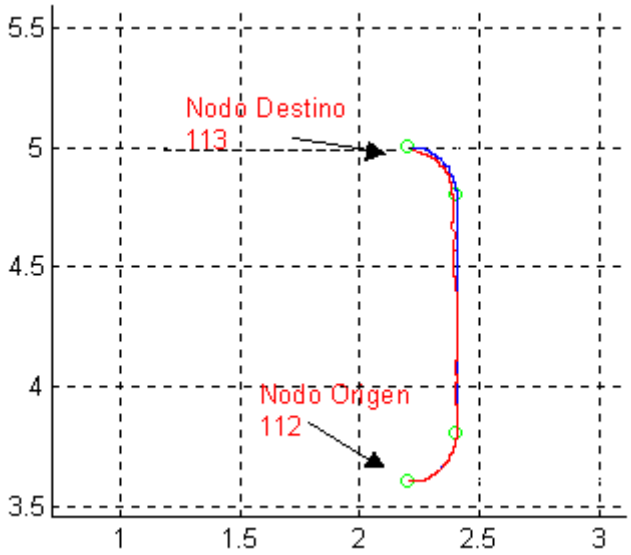


Figura 10.1. Recorrido de la silla de ruedas entre dos despachos de un mismo pasillo (coordenadas en metros).
Trazo rojo: Respuesta real del sistema.
Trazo azul: Respuesta planificada por el generador de trayectorias.

La figura 10.2 muestra el perfil de respuesta de las velocidades de avance (V) y de giro (Ω) que han hecho posible el recorrido. En la gráfica de V se puede observar el cambio en el perfil diseñado según se van ejecutando las distintas tareas de giro y de avance. Por otro lado, de la gráfica de Ω quizás llame la atención la aparente respuesta oscilatoria de la misma, que, sin embargo no se aprecia en el movimiento de la silla, y que se debe a los continuos cambios de orientación de la misma.

Además, en la misma figura 10.2, se muestra también el error de posición y de orientación del móvil, que como se ha comentado en el capítulo 8 tiene valores bastante aceptables (5.1cm de media de error de desplazamiento y -12.5 grados de error de orientación debido a los picos iniciales) siempre inapreciables para el usuario.

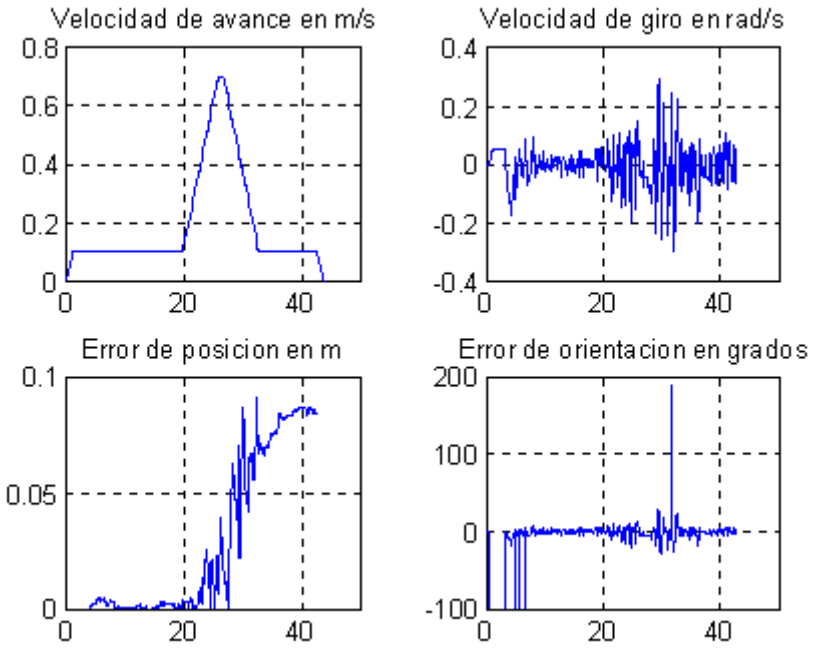


Figura 10.2. Gráficas de V,Ω,e_V y e_Ω frente al tiempo (s) para un recorrido entre dos despachos de un mismo pasillo.

En la figura 10.3 se muestra otro recorrido. En este caso se trata un avance en línea recta para acceder de un laboratorio de la primera planta a otro, localizado en el mismo pasillo también. Este ejemplo presenta interés desde el punto de vista de las molestas oscilaciones en las tareas de avance, que, tal y como se aprecia, son inexistentes.

Además, en la figura 10.4 se muestran también las gráficas de V, Ω y los errores de posición y orientación, donde, quizás lo más interesante sea observar el perfil trapezoidal de la velocidad de avance.

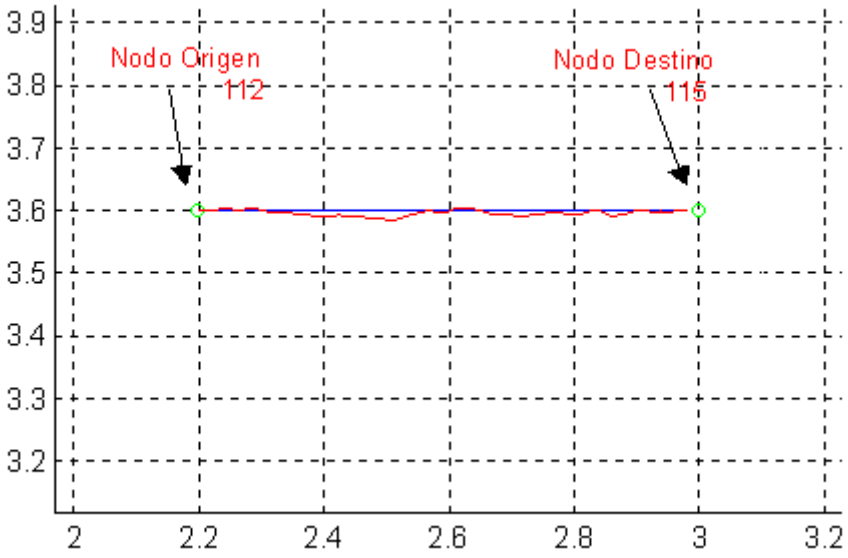


Figura 10.3. Recorrido de la silla de ruedas para un recorrido en línea recta (coordenadas en metros).
Trazo rojo: Respuesta real del sistema.
Trazo azul: Respuesta planificada por el generador de trayectorias.

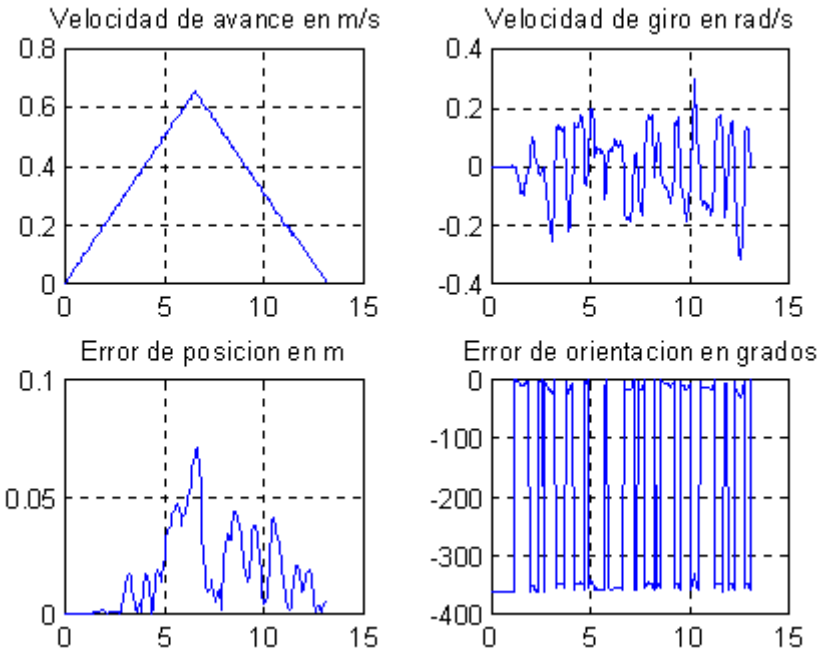


Figura 10.4. Gráficas de V,Ω,e_V y e_Ω frente al tiempo (s) para un recorrido en línea recta.

10.1.2. Recorridos largos.

En la figura 10.5 se muestra finalmente un recorrido más largo, el que describe la silla de ruedas para ir de un pasillo a otro dentro de una misma planta. En este caso se observa también como se traza el camino en los nodos de transición.

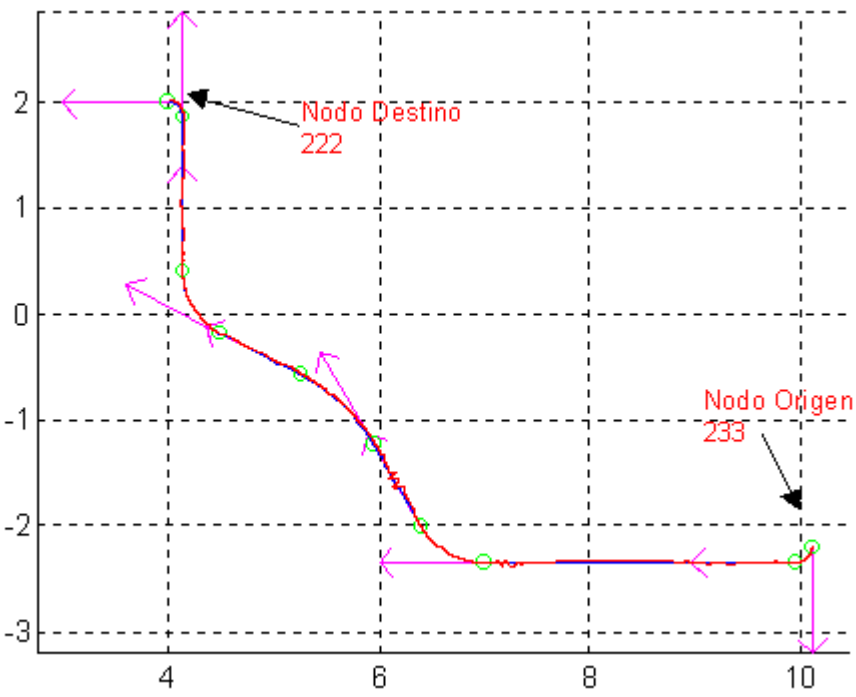


Figura 10.5. Recorrido de la silla de ruedas cambiando de pasillo.
(coordenadas en metros).
Trazo rojo: Respuesta real del sistema.
Trazo azul: Respuesta planificada por el generador de trayectorias

Las flechas mostradas indican la consigna de orientación en las distintas tareas afrontadas por el móvil. Se han incluido en este punto para facilitar la lectura del gráfico.

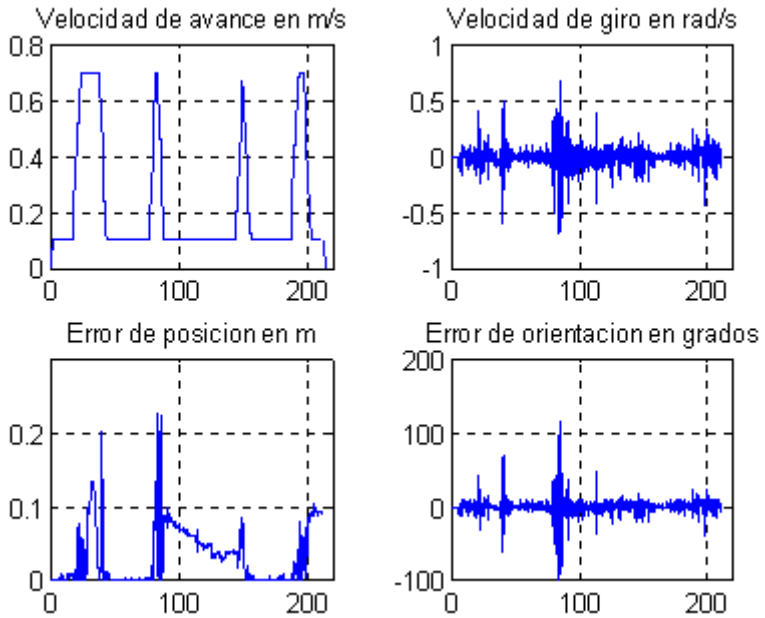


Figura 10.6. Gráficas de V,Ω,e_V y e_Ω frente al tiempo (s) para un recorrido cambiando de pasillo.

En la figura 10.6 se muestran las correspondientes consignas de V y Ω, con la evolución de sus respectivos errores.

En este último ejemplo el error parece mayor, ya que el móvil tiende a perderse en las tareas de avance, sin embargos e conserva una media semejante a casos anteriores (6.4cm de e_D y 0.5 grados de e_θ), y el movimiento sigue siendo confortable y la trayectoria perfectamente seguida.

10.2. Conclusiones y trabajos futuros.

En el apartado anterior se observa el correcto funcionamiento del sistema de navegación que en el primer capítulo de la memoria se plantea. Como conclusiones del trabajo se observa que el sistema diseñado es robusto para el entorno en el que se ha planteado, además de proporcionar una conducción cómoda para el discapacitado y aportar una solución interesante al problema del mapeado del entorno.

Quedan, sin embargo, muchas tareas abiertas a posteriores desarrollos y mejoras futuras que sería interesante añadir al sistema completo. A lo largo de la memoria se han ido planteado en los distintos módulos de navegación elementos que, si bien no son fundamentales para la tarea de navegación, sí facilitan el proceso completo. Los más importantes son los que se muestran a continuación.

En primer lugar es importante completar el sistema de posicionamiento absoluto del móvil con un módulo de visión, que permita corregir la posición obtenida por deadreckoning a través de las marcas localizadas en los nodos del edificio a recorrer. En el capítulo 4 se plantean las bases del sistema, si bien la implementación final requiere un gran trabajo de estudio y desarrollo que permita obtener en tiempo de ejecución las coordenadas reales de la silla con referencia a la marca. Este trabajo implica varias tareas de visión (reconocimiento, identificación y procesado) que, debido a la gran cantidad de datos proporcionados por una cámara, requerirían un procesador hardware externo al sistema, para que el resultado pudiese proporcionarse en tiempo real.

Un elemento que también ha sido ampliamente estudiado en el proyecto, si bien no se ha proporcionado la solución hardware sobre la que se implementa, es el transmisor de radio que permita a la silla capturar los mapas del entorno en movimiento. En los capítulos anteriores se establece el funcionamiento y la localización de los mismos, por lo que solo faltaría elegirlos y colocarlos para poder realizar pruebas de campo cambiando de mapa en la navegación. Este tema es otro importante y a la vez inmediato punto de trabajo sobre el que seguir avanzando.

Otro aspecto al que no se da solución en el proyecto es el de la detección de obstáculos, necesaria para moverse en un entorno cerrado como el presentado. En trabajos anteriores se han presentado distintas alternativas al problema de la localización, e incorporación de obstáculos a una ruta planificada. Todas ellas parten de un conjunto más o menos complejo de sensores, si bien generalmente siempre se basan en un sistema de visión, por lo que esta tarea podría ser incorporada también en el procesador hardware de localización antes comentado.

Por otro lado, un elemento que queda también un poco abandonado en el trabajo es el del interfaz de usuario. Si bien este tema no es fundamental para el sistema de navegación si parece interesante buscar un modulo que amplíe la información intercambiada entre el usuario y el sistema.

a) *En el sentido usuario-máquina*, el sistema más adecuado para introducir los comandos podría ser un reconocedor de voz, evitando así los problemas motores que afectan muchas veces también a las extremidades superiores del usuario. Otras opciones serían los pulsadores y los distintos módulos joystick que actúan como un puntero en una lista de selección que se presenta al usuario. Las posibilidades en este campo son muchas y dependen del usuario en concreto, por lo que tal y como se ha planteado la recepción de consignas, el sistema permite adaptar el módulo más adecuado en cada momento.

   Además el interfaz podría permitir al usuario elegir el camino bajo ciertas circunstancias, o al menos elegir entre varias rutas posibles.

b) *En sentido máquina-usuario*, sería interesante incorporar algún sistema que fuese informando al usuario del camino elegido por el navegador en cada momento así como de su evolución. En la propia pantalla del PC portátil, o en una pantalla de un PC empotrado se podrían presentar sobre el plano del propio edificio estos datos, junto a los mensajes de error que puedan aparecer a lo largo de la ejecución del movimiento, de modo que el usuario sepa lo que está ocurriendo en cada instante. Si bien esta parte no es necesaria da confianza a la persona que va sobre la silla, pues como ente inteligente desea saber donde es transportada.

Aunque estos son los aspectos más interesantes, muchos otros trabajos quedan abiertos a partir de este, pues como plataforma de navegación por interiores, el sistema desarrollado permite realizar pruebas de investigación en campo con gran facilidad:

- pruebas con otros tipos de controladores,
- elección de un núcleo operativo para sistemas en tiempo real que permita una óptima repartición de recursos,
- sistemas de codificación de planos automáticos para trabajar con la silla en otros entornos,

- mejora de la plataforma hardware, incorporando un sistema de alimentación adecuado y sustituyendo el ordenador portátil por un PC empotrado de menor consumo y más potencia,
- proceso de autolocalización para cuando el móvil se encuentra perdido,
- incorporar sensores de proximidad para movimientos precisos como el acceso a un ascensor, y diseñar procesos especiales de generación de trayectoria y control de la misma,
- incluir elementos de interacción con el edificio (mediante comunicación por radio o infrarrojos), para poder abrir puertas o actuar sobre ascensores [8], facilitando el movimiento de la silla,
- etc.

Por tanto, y como conclusión final, el proyecto desarrollado no pretende ser un trabajo cerrado o finalizado sino más bien una primera aportación en un área de investigación sobre la que se seguirá trabajando.

III. MANUAL DE USUARIO

En esta sección se muestran todos los pasos necesarios para poder poner en funcionamiento el sistema de navegación en entornos interiores para la silla de ruedas desarrollado en este proyecto.

La ejecución del navegador implementado requiere de la plataforma completa que se mostró en el capítulo 9 de la memoria: La silla de ruedas con su electrónica de bajo nivel, un PC portátil sobre el que se ejecuta la aplicación, cuyo puerto paralelo ha de soportar el protocolo EPP y la tarjeta de interfaz que permite intercomunicar ambos elementos.

1. El sistema de navegación.

Todo el algoritmo de navegación se incluye en un fichero ejecutable llamado *navega.exe.*

Para adaptarse a las distintas necesidades del sistema empleado, el programa diseñado se proporciona en versión de Windows y en versión DOS, si bien las pruebas realizadas en el desarrollo del mismo han sido siempre hechas sobre DOS, evitando el apropiamiento de recursos por parte del sistema operativo Windows.

Una vez instalado el ordenador portátil a bordo de la silla de ruedas solo es necesario arrancar la aplicación para iniciar el proceso de navegación. Para ello basta con ejecutar desde la línea de ordenes el programa con la siguiente sintaxis:

navega Nombre_Nodo_Origen Nombre_Nodo_Destino Nombre_Mapa

En caso de que la sintaxis sea incorrecta el sistema devuelve un mensaje de error como el siguiente:

Error!!! Sintaxis:
navega Nombre_Nodo_Origen Nombre_Nodo_Destino Nombre_Mapa

En caso contrario empieza a ejecutarse el programa de navegación, en el transcurso del cual se presentan distintos mensajes al usuario, informando de la evolución del proceso a la vez que la silla se mueve (como se va generando cada una de los tramos entre nodos y dentro de estos a su vez, cada una de las tareas de cada tramo).

En caso de producirse algún error, este también es indicado mediante el mensaje pertinente por la pantalla del PC. De este modo se informa de cuando la silla se ha desorientado sin posibilidad de recuperar la trayectoria, y de cuándo fallan las comunicaciones con el interfaz o en etapa de bajo nivel.

Evidentemente, todos estos mensajes tienen sentido solo en las tareas de desarrollo que se realicen sobre el sistema de navegación implementado, pues para el usuario final de la silla sería interesante diseñar un interfaz mas apropiado, tal y como se comentaba ya en el capítulo 10.

La herramienta de desarrollo se completa un algoritmo de presentación de resultados que permite visualizar mediante gráficas de Matlab datos sobre la evolución de la silla en su desplazamiento, una vez ha alcanzado el objetivo. Para poder implementar esta tarea de depuración es necesario almacenar datos durante el proceso de navegación. El software presentado en la sección de Planos no incluye estas funciones, pues son tareas que consumen mucho tiempo y no tienen utilidad en el proceso de navegación, sin embargo en el soporte software del libro del proyecto se incluye también lo que se ha venido a llamar *"versión de desarrollo"*, que incluye las funciones de almacenamiento de datos.

Los ficheros de datos generados con la versión de desarrollo son los siguientes:

- *trayecto.m* -> Almacena las coordenadas de los nodos que componen la ruta
- *sigueme.m* -> Con las coordenadas de los puntos que forman la trayectoria diseñada por generador.
- *movil.m* -> Contiene los puntos recorridos por el móvil en cada periodo de muestreo

- *consigna.m* -> Incluye el valor de las consignas de V (velocidad de avance) y Ω (velocidad de giro) generadas en cada periodo de muestreo por el controlador de posición.

Para presentar los datos en Matlab se emplean dos ficheros *script* que también se incluyen en el soporte informático anexo (mostrar.m y flachas.m).

Una vez finalizado la ruta pedida, el sistema la silla de ruedas se para y el programa de navegación finaliza, teniendo que volver a ejecutarlo para trazar un nuevo recorrido.

2. El sistema de bajo nivel y la tarjeta de interfaz.

Para poder poner en funcionamiento la silla de ruedas es necesario además realizar la conexión con el móvil y cargar el programa de bajo nivel en los nodos motores.

La conexión entre la tarjeta de comunicaciones y el PC se realiza mediante un cable paralelo de 25 hilos que une el DB25 que posee la tarjeta para tal efecto (ver figura 9.7) y el puerto paralelo del PC. Es importante realizar la comprobación de que el protocolo elegido para el puerto paralelo es EPP.

La tarjeta de comunicaciones posee además un conector de cable telefónico (RJ11) que se emplea para unir el nodo de comunicaciones al resto de la red LonWorks. En la figura III.1 se muestra una fotografía del sistema completo donde se aprecian las conexiones necesarias. De hecho el sistema es bastante simple, y para una posterior versión se puede fácilmente incluir el sistema de comunicaciones en el PC empotrado que ejecuta el algoritmo de navegación.

Figura III.1. Vista detallada del sistema de conexiones del navegador.

Por otro lado es necesario que los nodos tengan la aplicación correcta en memoria, por lo que se ha de realizar la carga de los mismos mediante el equipo de desarrollo LonBuilder. En el proyecto [5] se presenta paso a paso cual es el método de trabajo con esta herramienta.

Como las tarjetas de la red LonWorks (nodos de control de bajo nivel, y nodo de comunicaciones) reciben la alimentación de la batería de la silla a través de la propia red, el sistema global es totalmente autónomo durante alrededor de 2 horas.

IV. PLIEGO DE CONDICIONES

En esta sección se incluyen los aspectos técnicos de los equipos utilizados y los programas requeridos para la realización del proyecto.

1. Equipos físicos.

- *Ordenador portatil Pentium-120 (para la ejecución del navegador)*

Microprocesador	Pentium-Intel
Velocidad de reloj	120 MHz o mayor
Memoria RAM	16 Mbytes
Disco duro	No requiere mucha capacidad
Alimentación	GP280AFH 9.6V 2800mA/h

- *Ordenador Pentium-200 (para las tareas de depuración)*

Microprocesador	Pentium-Intel
Velocidad de reloj	200 MHz
Memoria RAM	16 Mbytes
Disco duro	1Gbyte
Monitor	SVGA-Color

- *Impresora HP Laserjet 2100*

Modo de impresión	Láser
Velocidad de páginas	16ppm
Buffer de entrada	4Mbytes
Comunicaciones	Serie y paralelo
Resolución	600ppp

- *Impresora HP Laserjet 4500 C*

Modo de impresión	Láser
Velocidad de páginas	8ppm
Buffer de entrada	4Mytes
Comunicaciones	Serie y paralelo
Resolución	600ppp (color)

- *Silla de ruedas GARANT del modelo 63E-PRO que incluye los siguientes elementos:*

Dos motores de tracción del tipo GP76.50Br0.4, cuya alimentación es de 24V. Incorpora un freno (electroimán) del tipo 73 341-05A10 de la marca BINDER, dos encóders de la casa comercial SIKO modelo IG16-0065 ABI/200/PP y montados sobre el eje del motor. Batería de alimentación de 24V para el sistema completo.

- *Equipo de desarrollo de redes LonWorks (LonBuilder), con los siguientes elementos:*

Network Manager (gestor de red), es el gestor central del equipo de desarrollo; Protocol Analycer (analizador de protocolo), que permite analizar la carga instantanea por la red, la tasa de error en envios, y otros parámetros asociados a la comunicacion LonWorks; Router, para realizar la descarga de programas a los nodos de aplicación, 2 emuladores, para tareas de depuración off-chip.

2. Equipos lógicos.

- *Sistema operativo Windows, y DOS*

Versiones MSDOS 6.22 y Windows95 y Windows98

- *MATLAB (versión 4.2b) y los siguientes Toolboxes:*

Simulink (versión 1.3)
Control System Toolbox

- *Borland C++ v.4.52.*
- *Compilador y depurador de NeuronC (proporcionado por Echelon)*
- *Procesador de textos (Word2000)*

V. PLANOS

En esta sección se incluyen los listados de todos los programas desarrollados en el proyecto, organizados de la siguiente forma:

1. Programas para el PC
2. Programas para el NeuronChip.

1. Programas para el PC.

Los programas para el PC se han desarrollado con el entorno de programación Borland C++ versión 4.52, y se han dividido en varios módulos:

- *gestor.c*: Es el programa principal, y realiza la gestión de tareas del sistema.
- *genera.c*: Contiene todas las funciones relacionadas con la generación de trayectorias.
- *controla.c*: Este módulo contiene las funciones del controlador de posición del sistema.
- *planruta.c*: Módulo de planificación de rutas o caminos.
- *driver.c*: Contiene las funciones de configuración y comunicación del EPP.

Todos los ficheros excepto *driver.c* tienen un fichero *.h* asociado. A continuación se incluyen todos estos módulos.

- *gestor.c*

```
/*****************************************************************/
/*                        Programa Principal                     */
/*                                                               */
/*                    Gestor de Trayectorias                     */
/*                   TFC Marta Marron. Navegacion                */
/*                                                               */
/*                     gestor.c             8/8/00               */
/*****************************************************************/
#include "gestor.h"

/*****************************************************************/
/*           Area de vbles globales de todo el proyecto          */
/*****************************************************************/
FILE *Mapa;       // puntero al fichero que contiene el mapa
int Ruta[MAX_NODOS_MAPA];     // lista de nodos que constituyen la ruta

typedef struct
{       float x;
        float y;
        float Theta;
}Posicion;
Posicion PosicionReal;

typedef struct{
        float xCentro;
        float yCentro;
        float Radio;
}Giro;

typedef struct{
        Posicion Origen;
        Posicion Destino;
        Giro Circulo;
        int Status;
}Trayectoria;

// para comunicar controller y generador
Trayectoria BufferTareas[MAX_TAREAS];        // buffer circular de tareas.
int NumTareaControl,NumTareaGenera;  // los punteros del buffer
int ObjetivoAlcanzado;


/*****************************************************************/
/*           Area de vbles globales en este fuente               */
/*****************************************************************/
static clock_t Contador;

// vbles solo para depuracion....
static long int Posx, Posy, PosTheta;
static int i=0;
static char NumeroFP[20];


/*****************************************************************/
/*           Area de vbles globales externas                     */
/*****************************************************************/

/*****************************************************************/
/*             Funciones                                         */
/*****************************************************************/
```

```
//---------------------------------------------------------------
// Nombre: Buscar_Nodo
// Funcion:
//     Permite buscar un nodo en el mapa activo. Devuelve toda la
//             informacion asociada al mismo.
// Parametros entrada:
//             int NombreNodo -> entero con el nombre jerarquico del mapa
// Parametros salida:
//             char * -> puntero a array con la informacion sobre el nodo
//---------------------------------------------------------------

char *Buscar_Nodo(int NombreNodo)
{
        char Nodo[MAX_LONG_NODO], *Nombre;
        int j;

        /* Ponemos el puntero del fichero al principio por si acaso */
        rewind(Mapa);
        j=0;

        /* buscando el nodo cuyo campo nombre coincida con el deseado */
        do     /* mientras no encuentre un nodo cuyo nombre coincida*/
        {
                fgets(Nodo,MAX_LONG_NODO,Mapa);
                Nombre = strtok(Nodo," ");
                                /* cojo un elemento del mapa, es el nombre */
        }while((NombreNodo!=atoi(Nombre))&&((j++)<=MAX_NODOS_MAPA));

        /* En la variable "nodo" tengo el nodo cuyo nombre coincide con nodo_Partida */
        if(j<MAX_NODOS_MAPA)    return (char *)Nodo;
        else
        {  puts("Nodo no encontrado");
                return(NULL);
        }

/* Fin de Funcion Bucar_Nodo */
}


//---------------------------------------------------------------
// Nombre: Llamando_Controlador
// Funcion:
//     Lleva conteo del tiempo, para llamar al controlador cada Ts
// Parametros entrada:
//                     Ninguno. Son vbles globales...
// Parametros salida:
//        Ninguno, no son necesarios (en ppio)
//---------------------------------------------------------------

void Llamando_Controlador(void)
{
        clock_t Aux;

        // si ha pasado un Ts se llama al controlador
        if((Aux=clock())>=(Contador+TS))
        {       fprintf(Proceso, "\nTs: %d", (Aux - Contador));
                Contador=Aux;
                Controla();
        }

        /* Fin de la funcion Llamando_Controlador */
}
```

```
/*****************************************************************/
/*                     programa principal                        */
/*****************************************************************/

//---------------------------------------------------------------
// Nombre: Gestor_Trayecto. es el MAIN del programa
// Funcion:
//     Realiza la gestion de trayectorias del sistema. Llama al
//     planificador de rutas, al generador de trayectorias y al controldor
//             Ademas controla el punto en que se encuentra el movil.
//             El es proceso que gestiona el tiempo de CPU mientras esta
//             andando la silla
//             Solamente se lleva a cabo un unico trayecto, luego la silla
//             se para. No se permite inicilamente cambio de destino en el
//             camino.
// Parametros entrada:
//                     Ninguno. Son vbles globales. (la ruta)
// Parametros salida:
//        Ninguno, no son necesarios (en ppio)
//---------------------------------------------------------------

void main(int argc,char *argv[])
{
       int NumRuta,Origen,Destino;
       long j;
       int InicioPosicion=1;

       //-------------------------------------------------------
       // Inicializaciones
       //-------------------------------------------------------

       // comprobando parametros y mensaje inicial del navegador
       if(argc!=4)
       {   printf("Error!!! Sintaxis: %s nombre_nodo_Partida nombre_nodo_destino
              fichero_mapa \n",argv[0]);
                     exit(-1);
       }
       puts("Planificador de Rutas. \nTFC Marta Marrón. Navegacion\n");

       // Preparando las comunicaciones, la DUALPORT
       initPort();           // Inicializa los puertos del modo EPP
       clearMemory(); // prueba la memoria

       // Inicializando parametros de generacion trayectorias.Para que encuentre
       // el final de la trayectoria.
       NumTareaControl=-1;  // no esta haciendo ni la 1º
       NumTareaGenera=0;            // la tarea que va a ocupar
       ObjetivoAlcanzado=0;  // se pondra a 1 cuando el controlador alcance
                                    // destino
       for(j=0;j<MAX_TAREAS;j++)
              BufferTareas[j].Status=-1;           // indica que esa posicion5 del
                                                   // buffer esta vacia


       //-------------------------------------------------------
       // 0º proceso.... Mapa del entorno: Abriendo el fichero que tiene el mapa
       //-------------------------------------------------------
       if(!(Mapa=fopen(argv[3],"r")))
       {  puts("Error Abriendo el mapa");
              exit(-1);
       }


       //-------------------------------------------------------
```

```
	// 1º proceso.... Planificador de Caminos: llamando al planificador para
	// obtener la ruta
	//-----------------------------------------------------
	puts("/* Llamando al planificador de caminos */\n");
	Plan_Ruta(argv[1], argv[2]);
	Origen=atoi(strtok(argv[1],"_"));
	Destino=atoi(strtok(argv[2],"_"));


	//-----------------------------------------------------
	// 2º y 3º proceso.... controller y generador.
	//-----------------------------------------------------
	puts("/* Llamando al controlador y al generador */\n");
	NumRuta=0;
	Contador=clock();

	// AQUI EMPIEZA EL BUCLE INFINITO QUE LLAMA CONTINUAMENTE AL CONTROLADOR
	do{
		// 1º se mira a ver si hay hueco en el buffer para meter tareas de
		// nuevo tramo generado, al menos 4 tareas
		if((Ruta[NumRuta]!=Destino)&&(HAY_ESPACIO))
		{  // generando nuevo trayecto
			Genera_Trayecto(Ruta[NumRuta],Ruta[NumRuta+1]);
			NumRuta++;
		}

		// 2º se comprueba si no se ha inicializado aun la posicion, pues
		// es la 1º vez
		if(InicioPosicion)
		{	// Va a ser la posicion del nodo de partida
			PosicionReal.x=BufferTareas[0].Origen.x;
			PosicionReal.y=BufferTareas[0].Origen.y;
			PosicionReal.Theta=BufferTareas[0].Origen.Theta;
			InicioPosicion=0;
		}

		// 3º si es el ultimo trayecto se indica al controlador
		if(Ruta[NumRuta]==Destino)
			BufferTareas[NumTareaGenera].Status=FINAL;

		// 4º veo si ha pasado el periodo, y si es asi se llama al
		// controller
		Llamando_Controlador();

	}while(!ObjetivoAlcanzado);

	//-----------------------------------------------------
	// 4º proceso.... finalizando resultados,
	//-----------------------------------------------------

	// como en principio solo hay un mapa se cierra al salir.
	fclose(Mapa);

	/* Fin de la funcion principal main */
}
```

- *gestor.h*

```
/*****************************************************************/
/*      Programa Principal. Fichero cabecera                     */
/*      TFC Marta Marron. Navegacion                             */
/*                                                               */
/*      gestor.h              8/4/00                             */
/*****************************************************************/

/*****************************************************************/
/*                    Area de librerias                          */
/*****************************************************************/
#include <string.h>
#include <math.h>
#include <stdio.h>
#include <mem.h>
#include <stdlib.h>
#include <time.h>

/*****************************************************************/
/*                    Area de constantes                         */
/*****************************************************************/

// ctes de tamaños varios.
#define MAX_NODOS_MAPA        200
#define MAX_LONG_NODO 200
#define MAX_LONG_NOMBRE       3
#define MAX_NODOS_RUTA        50
#define MAX_TAREAS            50
#define MAX_X                 MAX_TAREAS*MAX_NODOS_RUTA*2*10
#define MAX_XC                20000


// ctes de buffer tareas
#define FINAL                         0      // indica es la ultima tarea
#define HAY_ESPACIO           ((BufferTareas[NumTareaGenera].Status==-1)&&
                              (BufferTareas[NumTareaGenera+1].Status==-1)&&
                              (BufferTareas[NumTareaGenera+2].Status==-1)&&
(BufferTareas[NumTareaGenera+3].Status==-1)&&(BufferTareas[NumTareaGenera+4].Status==-1))

// otras constantes
#define ESCALA                        500
#define TS                            50$
```

- *genera.c*

```
/*****************************************************************/
/*            Generador de Trayectorias                          */
/*             TFC Marta Marron. Navegacion                      */
/*                                                               */
/*             genera.c              8/8/00                      */
/*****************************************************************/
#include "genera.h"

/*****************************************************************/
/*               Area de vbles globales a este fuente            */
/*****************************************************************/
static float ConsignaGiro[3];
static float ConsignaAvance[3];

/*****************************************************************/
/*         Area de vbles globales en todo el proyecto            */
/*****************************************************************/

/*****************************************************************/
/*               Area de vbles globales EXTERNAS                 */
/*****************************************************************/
// Declaradas en  genera.h
//extern FILE *Mapa;
//extern Trayectoria BufferTareas[MAX_TAREAS];
//extern int NumTareaGenera;

/*****************************************************************/
/*             Funciones                                         */
/*****************************************************************/
//---------------------------------------------------------------
// Nombre: Tarea_Giro4
// Funcion:
//     Tarea que programa el controlador para avanzar en linea recta
//             hasta el destino.
// Parametros entrada:
//     loat Parametros -> coordenadas y vbles necesarias para el calculo:
//     Parametros[1]= x punto acceso nodo sobre el que giro;
//     Parametros[2]= y punto acceso nodo sobre el que giro;
//     Parametros[3]= x punto acceso nodo cercano;
//     Parametros[4]= y punto acceso nodo cercano;
//     Parametros[5]= Orientacion de normal al nodo;
//     Parametros[6]= Distancia al punto de acceso al nodo;
//     Parametros salida:
//     No tiene. solo completa la estructura de la trayectoria construida
//    sobre trayectoria actual
/*---------------------------------------------------------------*/

void Tarea_Giro(float *Parametros)
{
       double Aux;
       double OrientaNormal, DistanciaAccesoNodo, xNodo1, yNodo1, xNodo2,
       yNodo2;
       double OrientaConsigna, xConsigna, yConsigna;

       //-------------------------------------------
       // Lo primero es recoger los parametros para que el codigo sea legible
       //-------------------------------------------
       xNodo1= Parametros[0];
       yNodo1= Parametros[1];
       xNodo2= Parametros[2];
       yNodo2= Parametros[3];
```

```
OrientaNormal= Parametros[4];
DistanciaAccesoNodo= Parametros[5];

//-------------------------------------------
// Ahora hay que calcular la direccion consigna
//-------------------------------------------
// Eliminando indeterminaciones.
if((xNodo2-xNodo1)==0)
{      if((yNodo2-yNodo1)>0) OrientaConsigna=90;
       else if((yNodo2-yNodo1)<0)    OrientaConsigna=270;
       // en este caso el punto es el mismo. Seguir recto, la normal de
       // cualq. de dos nodos!
       else OrientaConsigna=OrientaNormal;
}

// en proceso normal....
else
{      OrientaConsigna=atan((yNodo2-yNodo1)/(xNodo2-xNodo1));
       OrientaConsigna=(OrientaConsigna*360)/(2*PI);
       if(OrientaConsigna<0) OrientaConsigna+=360;
       OrientaConsigna=fmod(OrientaConsigna,360);
}

// Ahora se comprueba que no estemos en el 2º o 3º cuadrante. Hay que
// sumar 180
if((xNodo2-xNodo1)<0)
{      OrientaConsigna+=180;
       OrientaConsigna=fmod(OrientaConsigna,360);
}

//-------------------------------------------
// Ahora hay que calcular la consigna de posicion
//-------------------------------------------
//Elimino indet. de Ay=0
if(yNodo2==yNodo1)
{      yConsigna=yNodo1;            // Es que coinciden yP1 e yP2;
       if((xNodo2-xNodo1)==0)xConsigna=xNodo1 //...y tambien xP1 e yP1...
                             // hay que seguir recto
       else xConsigna=xNodo1+((fabs(xNodo2-xNodo1))/(xNodo2-
       xNodo1))*DistanciaAccesoNodo; // toda la distancia se da en x
}

// en caso normal....
else if((yNodo2-yNodo1)>0)
{      Aux=(((xNodo2-xNodo1)*(xNodo2-xNodo1))/((yNodo2-yNodo1)*(yNodo2-
              yNodo1)))+1;
       Aux=sqrt(Aux);
       yConsigna=(DistanciaAccesoNodo/Aux)+yNodo1;
       xConsigna=((xNodo2-xNodo1)/(yNodo2-yNodo1))*(yConsigna-
                     yNodo1)+xNodo1;
}

//   y si es negativo....
else if((yNodo2-yNodo1)<0)
{      Aux=(((xNodo2-xNodo1)*(xNodo2-xNodo1))/((yNodo2-yNodo1)*(yNodo2-
              yNodo1)))+1;
       Aux=sqrt(Aux);
       yConsigna=(-DistanciaAccesoNodo/Aux)+yNodo1;
       xConsigna=((xNodo2-xNodo1)/(yNodo2-yNodo1))*(yConsigna-
                     yNodo1)+xNodo1;
}

//-------------------------------------------
// fijando resultados-
```

```
	//-----------------------------------------------
	ConsignaGiro[0]=(float)OrientaConsigna;
	ConsignaGiro[1]=(float)xConsigna;
	ConsignaGiro[2]=(float)yConsigna;

	/*	Fin de la funcion Trayecto_Gira	*/
}

//---------------------------------------------------------------
// Nombre: Parametros_Giro
// Funcion:
//	Permite obtener los parametros de la trayectoria de giro
//		que necesitara el controlador (radio y centro de circunferencia)
// Parametros entrada:
//		Ninguno, todos son globales
// Parametros salida:
//		Ninguno, es global el array de tareas.
/*---------------------------------------------------------------*/

void Parametros_Giro(void)
{
	float m1,m2,b1,b2;
	float xCentro, yCentro,R;

	//-----------------------------------------
	//  obtengo el origen de la circunferencia
	//-----------------------------------------
	// en caso normal....
	if((BufferTareas[NumTareaGenera].Origen.Theta!=0) &&
		(BufferTareas[NumTareaGenera].Origen.Theta!=180) &&
		(BufferTareas[NumTareaGenera].Destino.Theta!=0) &&
		(BufferTareas[NumTareaGenera].Destino.Theta!=180))
	{
		if((BufferTareas[NumTareaGenera].Origen.Theta==90) ||
		(BufferTareas[NumTareaGenera].Origen.Theta==270))
			m1=0;
		else
		{	m1=tan(BufferTareas[NumTareaGenera].Origen.Theta*PI/180);
			m1=-1/m1;   // la normal
		}

		if((BufferTareas[NumTareaGenera].Destino.Theta==90) ||
		(BufferTareas[NumTareaGenera].Destino.Theta==270))
			m2=0;
		else
		{	m2=tan(BufferTareas[NumTareaGenera].Destino.Theta*PI/180);
			m2=-1/m2;   // la normal
		}

		b1=BufferTareas[NumTareaGenera].Origen.y-
		m1*BufferTareas[NumTareaGenera].Origen.x;
		b2=BufferTareas[NumTareaGenera].Destino.y-
		m2*BufferTareas[NumTareaGenera].Destino.x;
		xCentro=(b1-b2)/(m2-m1);
		// nunca las pendientes pueden salir iguales, error del generador
		yCentro=m1*xCentro+b1;
		// por ello nunca ha de dar divicion por 0
	}

	// la pendiente de la normal al punto de origen es infinito
	else if((BufferTareas[NumTareaGenera].Destino.Theta!=0)&&
		(BufferTareas[NumTareaGenera].Destino.Theta!=180))
	{
		xCentro=BufferTareas[NumTareaGenera].Origen.x;
```

```
                if((BufferTareas[NumTareaGenera].Destino.Theta==90) ||
                (BufferTareas[NumTareaGenera].Destino.Theta==270))
                        m2=0;
                else
                {       m2=tan(BufferTareas[NumTareaGenera].Destino.Theta*PI/180);
                        m2=-1/m2;   // la normal
                }

                b2=BufferTareas[NumTareaGenera].Destino.y-
                m2*BufferTareas[NumTareaGenera].Destino.x;
                yCentro=m2*xCentro+b2;
        }

        // la pendiente de la normal al punto de destino es infinito
        else if((BufferTareas[NumTareaGenera].Origen.Theta!=0) &&
                (BufferTareas[NumTareaGenera].Origen.Theta!=180))
        {
                xCentro=BufferTareas[NumTareaGenera].Destino.x;

                if((BufferTareas[NumTareaGenera].Origen.Theta==90) ||
                        (BufferTareas[NumTareaGenera].Origen.Theta==270))
                        m1=0;
                else
                {       m1=tan(BufferTareas[NumTareaGenera].Origen.Theta*PI/180);
                        m1=-1/m1;   // la normal
                }
                b1=BufferTareas[NumTareaGenera].Origen.y-
                        m1*BufferTareas[NumTareaGenera].Origen.x;
                yCentro=m1*xCentro+b1;
        }

        // falta por observar que ambas sean infinitas.....¿puede dar prblems?

        //-----------------------------------------
        // Finalmente hallo el radio
        //-----------------------------------------
        R=pow((BufferTareas[NumTareaGenera].Origen.x-
                xCentro),2)+pow((BufferTareas[NumTareaGenera].Origen.y-yCentro),2);
        R=sqrt(R);

        //-----------------------------------------
        // añadiendo parametros de giro
        //-----------------------------------------
        BufferTareas[NumTareaGenera].Circulo.xCentro=xCentro;
        BufferTareas[NumTareaGenera].Circulo.yCentro=yCentro;
        BufferTareas[NumTareaGenera].Circulo.Radio=R;

        /* Fin de la funcion Parametros_Giro */
}

//---------------------------------------------------------------
// Nombre: Genera_Trayecto
// Funcion:
//      Divide la tarea de mover la silla entre dos nodos, en subtareas
//              Programa las consignas de controlador, controla si se han cumplido
//              objetivos, e informa al gestor.
// Parametros entrada:
//              int Origen -> nombre del nodo de origen
//              int Destino -> nombre del nodo de destino
// Parametros salida:
//              Ninguno, es global el array de tareas.
/*---------------------------------------------------------------*/
```

```
void Genera_Trayecto(int Origen, int Destino)
{
        static Posicion FinTrayectoAnterior;
        char *NodoOrigen;
        char *NodoDestino;
        int HayGiro, AvancePrimera;
        int Origen1,Destino1,Origen2, Destino2, Origen3, Destino3;
        float xDestino, yDestino, xOrigen, yOrigen, OrientaOrigen,
        OrientaDestino, DistanciaNodoDestino, DistanciaNodoOrigen,
        OrientaDestinoFin;
        float xP1, yP1, xP2, yP2;
        float Parametros[6];

        // vbles auxiliares
        double Aux1,Aux2;

        //-------------------------------------------------------------
        // Inicializacion
        //-------------------------------------------------------------

        // reseteo los flags
        HayGiro=0;
        AvancePrimera=0;

        // desmenuzando el origen y el destino
        Origen1 = Origen%100;
        Origen2= Origen1/10;
        Origen3= Origen1%10;
        Destino1 = Destino%100;
        Destino2= Destino1/10;
        Destino3= Destino1%10;

        //.------------------------------------------------------------
        // 0 - Obtengo informacion sobre los nodos de origen y destino
        // Ademas calculo ya de paso los puntos de aproxc. a nodo P1 y P2
        //.------------------------------------------------------------

        // 1º recojo informacion sobre el nodo de origen del mapa y calculando
        // punto acceso.
        NodoOrigen=(char *)Buscar_Nodo(Origen);

        // Si no es un nodo fantasma/transicion
        if(Origen>=0)
        {       // recojo info asociada al nodo
                xOrigen=atof(strtok('\0'," "));      // Obtengo el valor del x del nodo
                                                     // origen
                yOrigen=atof(strtok('\0'," "));      // Obtengo el valor del y del nodo
                                                     // origen
                OrientaOrigen=atof(strtok('\0'," "));       // Obtengo la orientacion
                                                            // de la normal en grados
                DistanciaNodoOrigen=atof(strtok('\0'," "));
                                        // Distancia de aprox al nodo origen

                // cambiando la orientacion si es jerarquico, en funcion de
                // sentido avance
                // Si entra en jerarquia nivel 2 es sentido contrario al que dice
                // el mapa
                if(ORIGEN_JERARQUICO2 && VA_JERARQUIA2)
        OrientaOrigen=fmod(OrientaOrigen+180,360);
                // Si entra en jerarquia nivel 1 es sentido ES TAMBIEN contrario
                // al que dice el mapa
                else if(ORIGEN_JERARQUICO1 && VA_JERARQUIA1)
        OrientaOrigen=fmod(OrientaOrigen+180,360);
```

```
        // Obteniendo el punto de acceso al nodo
        xP1=xOrigen+DistanciaNodoOrigen*cos((2*PI/360)*OrientaOrigen);
        yP1=yOrigen+DistanciaNodoOrigen*sin((2*PI/360)*OrientaOrigen);
}

// si es un nodo fantasma hay que calcular el punto de acceso de otro modo
else
{       // entonces el nodo de origen es el que fue destino antes.
        xOrigen=FinTrayectoAnterior.x;
        yOrigen=FinTrayectoAnterior.y;
        OrientaOrigen=FinTrayectoAnterior.Theta;

        // el punto de acceso es el propio nodo fantasma, sobre el se gira
        xP1=atof(strtok('\0'," "));
                        //Obtengo el valor x del nodo origen (fantasma)
        yP1=atof(strtok('\0'," "));
                        //Obtengo el valor y del nodo origen (fantasma)

        // la distancia al nodo queda igual, pues se hizo eso en origen
        DistanciaNodoOrigen=atof(strtok('\0'," "));
                        //Distancia de aprox al nodo origen
}

// 2º recojo informacion sobre el nodo de destino del mapa y calculo el
// punto de acceso.
NodoDestino=(char *)Buscar_Nodo(Destino);
xDestino=atof(strtok('\0'," "));
                        //Obtengo el valor del x del nodo destino
yDestino=atof(strtok('\0'," "));
                        //Obtengo el valor del y del nodo destino

// Si no es un nodo fantasma/transicion
if(Destino>=0)
{       // recojo info asociada al punto de acceso
        // No le sumo 180 grados a la orientacion pues me viene bien para
        // los calculos, aunque sea entrada
        OrientaDestino=atof(strtok('\0'," "));
                                //Obtengo la orientacion de la normal en grados
        DistanciaNodoDestino=atof(strtok('\0'," "));
                                //Distancia de aprox al nodo origen

        // cambiando orientacion si es un nodo jerarquico en funcion de si
        // salgo o entro
        if(DESTINO_JERARQUICO1 && VIENE_JERARQUIA1)
        // viene de su jerarquia
        {       OrientaDestino=OrientaDestino-180;
                OrientaDestino=fmod(OrientaDestino+360,360);
                // para que no salga negativo
        }
        else if(DESTINO_JERARQUICO2 && VIENE_JERARQUIA2)
                // viene de su jerarquia
        {       OrientaDestino=OrientaDestino-180;
                OrientaDestino=fmod(OrientaDestino+360,360);
                // para que no salga negativo
        }

        // Obteniendo el punto de acceso al nodo
        xP2=xDestino+DistanciaNodoDestino*cos((2*PI/360)*OrientaDestino);
        yP2=yDestino+DistanciaNodoDestino*sin((2*PI/360)*OrientaDestino);
}

// si es un nodo fantasma hay que calcular el punto de acceso de otro modo
else
{       //obtengo la direccion del nodo destino eliminando indeterminaciones
```

```
        if((xDestino-xP1)==0)
        {       if((yDestino-yP1)>0)  OrientaDestino=90;
                else if((yDestino-yP1)<0)     OrientaDestino=270;
                // en este caso el punto es el mismo. Seguir recto, la
                // normal de cualq. de dos nodos!
                else OrientaDestino=OrientaOrigen;
        }

        // en caso normal....
        else
        {       OrientaDestino=atan((yDestino-yP1)/(xDestino-xP1));
                OrientaDestino=(OrientaDestino*360)/(2*PI);
                if(OrientaDestino<0)          OrientaDestino+=360;
                OrientaDestino=fmod(OrientaDestino,360);
        }

        // Ahora se comprueba que no estemos en el 2º o 3º cuadrante. Hay
        // que sumar 180
        if((xDestino-xP1)<0)
        {       OrientaDestino+=180;
                OrientaDestino=fmod(OrientaDestino,360);
        }

        // obtengo la distancia a destino
        DistanciaNodoDestino=atof(strtok('\0'," "));
                              //Distancia de aprox al nodo origen

        // Obteniendo el nodo destino real
        xDestino=xDestino-
                DistanciaNodoDestino*cos((2*PI/360)*OrientaDestino);
        yDestino=yDestino-
                DistanciaNodoDestino*sin((2*PI/360)*OrientaDestino);

        // Obteniendo el punto de acceso
        xP2=xDestino-DistanciaNodoDestino*cos((2*PI/360)*OrientaDestino);
        yP2=yDestino-DistanciaNodoDestino*sin((2*PI/360)*OrientaDestino);
}

//.-----------------------------------------------------------
// 1 - Si es movimiento entre ascensores
// Tengo que hacerle entrar, tarea solo hasta P*
//      y luego tengo que hacerle salir. Tarea de avance hasta P1
//.-----------------------------------------------------------

if(ORIGEN_ASCENSOR && DESTINO_ASCENSOR)
{// SE TRATA DE DOS TAREAS DE AVANCE, UNA DE ENTRADA Y UNA DE SALIDA.

        //      1º calculo punto de acceso al ascensor.
        ConsignaAvance[0]=FinTrayectoAnterior.Theta;
        ConsignaAvance[1]=xOrigen+DistanciaNodoOrigen*cos((2*PI/360)*
                              FinTrayectoAnterior.Theta);
        ConsignaAvance[2]=yOrigen+DistanciaNodoOrigen*sin((2*PI/360)*
                              FinTrayectoAnterior.Theta);

        // y actualizando el area para el controlador.
        BufferTareas[NumTareaGenera].Origen.x=FinTrayectoAnterior.x;
        BufferTareas[NumTareaGenera].Origen.y=FinTrayectoAnterior.y;
        BufferTareas[NumTareaGenera].Origen.Theta=FinTrayectoAnterior.Theta;
        BufferTareas[NumTareaGenera].Destino.x=ConsignaAvance[1];
        BufferTareas[NumTareaGenera].Destino.y=ConsignaAvance[2];
        BufferTareas[NumTareaGenera].Destino.Theta=ConsignaAvance[0];
                                      // HA DE COINCIDIR CON LA DE ORIGEN
        BufferTareas[NumTareaGenera].Status=AVANCE;
        NumTareaGenera=(NumTareaGenera+1)%MAX_TAREAS;
```

```
        // 2º Espero a que llegue al nuevo piso.
        puts("Silla en ascensor, esperando alcanzar nuevo piso. Pulse
               cuando ocurra");
        getch();

        // 3º salgo del ascensor, solamente hasta el nodo-ascensor
        ConsignaAvance[1]=xDestino;
        ConsignaAvance[2]=yDestino;
        ConsignaAvance[0]=fmod(OrientaDestino+180,360);
        // Actualizando el area para el controlador.
        BufferTareas[NumTareaGenera].Origen.x=BufferTareas[NumTareaGenera-
                       1].Destino.x;
        BufferTareas[NumTareaGenera].Origen.y=BufferTareas[NumTareaGenera-
                       1].Destino.y;
        BufferTareas[NumTareaGenera].Origen.Theta=BufferTareas[NumTareaGenera-
                       1].Destino.Theta;
        BufferTareas[NumTareaGenera].Destino.x=ConsignaAvance[1];
        BufferTareas[NumTareaGenera].Destino.y=ConsignaAvance[2];
        BufferTareas[NumTareaGenera].Destino.Theta=ConsignaAvance[0];
        BufferTareas[NumTareaGenera].Status=AVANCE;
        NumTareaGenera=(NumTareaGenera+1)%MAX_TAREAS;

        // y rertorno....
        FinTrayectoAnterior=(Posicion)BufferTareas[NumTareaGenera].Destino;
        return;
}

//.-------------------------------------------------------
// 2 - Para el resto de los casos
// TENGO QUE HACER:
// [TAREA GIRO (entre nodo y P1)]  + TAREA AVANCE (entre P1 y P2)
//                    [+ TAREA GIRO (entre P2 y nodo)]
//.-------------------------------------------------------

else
{  // 1º Tarea de giro:
        // el movil parte de la posicion del nodo y he de calcular
        // xydestino, OrientaDestino para lo cual necesita:
        // xyorigen, xyP1, xyP2, Distancia, Orienta -> todo al array de
        // parametros
        // devuelve las consignas en ConsignaGiro

        Parametros[0]= xP1;
        Parametros[1]= yP1;
        Parametros[2]= xP2;
        Parametros[3]= yP2;
        Parametros[4]= OrientaOrigen;
        Parametros[5]= DistanciaNodoOrigen;
        Tarea_Giro(Parametros);

        // Solo si la orientacion de salida del nodo NO coincide con la
        // consigna
        // pedida por el tramo de giro es necesario girar.
        // por problemas de resolucion de floats...
        Aux1=((double)((long int)(ConsignaGiro[0]*100)))/100;
        Aux2=((double)((long int)(OrientaOrigen*100)))/100;
        if(Aux1!=Aux2)
        {       //puts("** fue necesario tarea giro inicial");

                // y actualizando el area para el controlador.
                BufferTareas[NumTareaGenera].Origen.x=xOrigen;
                BufferTareas[NumTareaGenera].Origen.y=yOrigen;
```

```
                        BufferTareas[NumTareaGenera].Origen.Theta=OrientaOrigen;
                        BufferTareas[NumTareaGenera].Destino.x=ConsignaGiro[1];
                        BufferTareas[NumTareaGenera].Destino.y=ConsignaGiro[2];
                        BufferTareas[NumTareaGenera].Destino.Theta=ConsignaGiro[0];
                        BufferTareas[NumTareaGenera].Status=GIRO;

                        // añadiendo parametros de giro
                        Parametros_Giro();
                        NumTareaGenera=(NumTareaGenera+1)%MAX_TAREAS;

                }

                else
                {       //puts("** no fue necesario tarea de giro inicial");
                        AvancePrimera=1;
                }

                // 2º Tarea de avance
                // El movil ha de terminar en la misma posicion que necesita como
                // entrada la tarea de giro

                // la orientacion de salida es la de entrada
                ConsignaAvance[0]=ConsignaGiro[0];

                // calculo la orientacion del nodo destino.. SOLO SI NO ES NODO DE
                // TRANSICION
                if(Destino>0)         OrientaDestinoFin=fmod(OrientaDestino+180,360);
                                // siempre hay que sumar 180 excepto....
                else OrientaDestinoFin=OrientaDestino;

// si no es asi es necesario dejar al movil en xynodo. nueva tarea de giro
// ahora sobre el nodo de destino, con punto auxiliar P1 en vez de P2
                // por problemas de resolucion de los float.
                Aux1=((double)((long int)(ConsignaAvance[0]*100)))/100;
                Aux2=((double)((long int)(OrientaDestinoFin*100)))/100;
                if(Aux1!=Aux2)
                {  Parametros[0]= xP2;
                        Parametros[1]= yP2;
                        Parametros[2]= xP1;
                        Parametros[3]= yP1;
                        Parametros[4]= OrientaDestinoFin;
                        Parametros[5]= DistanciaNodoDestino;
                        Tarea_Giro(Parametros);

                        ConsignaAvance[1]=ConsignaGiro[1];
                        ConsignaAvance[2]=ConsignaGiro[2];

                        HayGiro=1;
                }

                // Si la orientacion alcanzada en el tramo de giro coincide con la
                // de entrada al nuevo nodo NO hay que generar un nuevo giro al
                // llegar al final del tramo de avance. el destino es el nodo
                else
                {       //puts("** no fue necesaria tarea de giro final");

                        ConsignaAvance[1]=xDestino;
                        ConsignaAvance[2]=yDestino;
                }

                // y actualizando el area para el controlador.
                // el origen de esta tarea depende de cual fue la tarea anterior
                if(AvancePrimera)
                {       BufferTareas[NumTareaGenera].Origen.x=xOrigen;
```

```
                        BufferTareas[NumTareaGenera].Origen.y=yOrigen;
                        BufferTareas[NumTareaGenera].Origen.Theta=OrientaOrigen;
                        AvancePrimera=0;
                }
                else
                {       BufferTareas[NumTareaGenera].Origen.x=
                        BufferTareas[NumTareaGenera-1].Destino.x;
                        BufferTareas[NumTareaGenera].Origen.y=
                        BufferTareas[NumTareaGenera-1].Destino.y;
                        BufferTareas[NumTareaGenera].Origen.Theta=
                        BufferTareas[NumTareaGenera-1].Destino.Theta;
                }
                BufferTareas[NumTareaGenera].Destino.x=ConsignaAvance[1];
                BufferTareas[NumTareaGenera].Destino.y=ConsignaAvance[2];
                BufferTareas[NumTareaGenera].Destino.Theta=ConsignaAvance[0];
                BufferTareas[NumTareaGenera].Status=AVANCE;
                NumTareaGenera=(NumTareaGenera+1)%MAX_TAREAS;

                // 2º Tarea de giro: SI HA HABIDO GIRO PARA LLEGAR A NODO DESTINO
                // El movil ha de terminar en la misma posicion que necesita como
                // entrada la tarea de giro

                if(HayGiro)
                {       //puts("** fue necesaria tarea de giro final");

                        ConsignaGiro[2]=yDestino;
                        ConsignaGiro[1]=xDestino;
                        ConsignaGiro[0]=OrientaDestinoFin;  //fmod(OrientaDestino+180,360);

                        HayGiro=0;

                        // y actualizando el area para el controlador.
                        BufferTareas[NumTareaGenera].Origen.x=
                                        BufferTareas[NumTareaGenera-1].Destino.x;
                        BufferTareas[NumTareaGenera].Origen.y=
                                        BufferTareas[NumTareaGenera-1].Destino.y;
                        BufferTareas[NumTareaGenera].Origen.Theta=
                                BufferTareas[NumTareaGenera-1].Destino.Theta;
                        BufferTareas[NumTareaGenera].Destino.x=ConsignaGiro[1];
                        BufferTareas[NumTareaGenera].Destino.y=ConsignaGiro[2];
                        BufferTareas[NumTareaGenera].Destino.Theta=ConsignaGiro[0];
                        BufferTareas[NumTareaGenera].Status=GIRO;

                        // añadiendo parametros de giro
                        Parametros_Giro();
                        NumTareaGenera=(NumTareaGenera+1)%MAX_TAREAS;

                }

                // y retornamos. guardando el ultimo movimento para ascensores
                FinTrayectoAnterior=(Posicion)BufferTareas[NumTareaGenera].Destino;
                return;
        }

}
```

- *genera.h*

```
/*****************************************************************/
/*             Generador de Trayectorias. Fichero cabecera       */
/*          TFC Marta Marron. Navegacion                         */
/*                                                               */
/*             genera.h         8/8/00                           */
/*****************************************************************/

/*****************************************************************/
/*                           Area de librerias                   */
/*****************************************************************/
#include <string.h>
#include <math.h>
#include <stdio.h>
#include <time.h>

/*****************************************************************/
/*                           Area de constantes                  */
/*****************************************************************/
// ctes varias
#define PI                          3.1415926

// ctes de la jerarquia
#define ORIGEN_ASCENSOR ((Origen2==0) && ((Origen3==1) || (Origen3==2) ||
(Origen3==6) || (Origen3==5)))
#define DESTINO_ASCENSOR ((Destino2==0) && ((Destino3==1) || (Destino3==2) ||
(Destino3==6) || (Destino3==5)))

#define ORIGEN_JERARQUICO2 ((Origen3==0)&&(Origen2!=0))
#define VA_JERARQUIA2 (Destino2==Origen2)
#define ORIGEN_JERARQUICO1 ((Origen2==0)&&((Origen3==7)||(Origen3==0)))
#define VA_JERARQUIA1 ((Destino3!=6)&&(Destino3!=1))

#define VIENE_JERARQUIA1 ((Origen3!=6)&&(Origen3!=1))
#define DESTINO_JERARQUICO1 ((Destino2==0)&&((Destino3==7)||(Destino3==0)))
#define VIENE_JERARQUIA2 (Destino2==Origen2)
#define DESTINO_JERARQUICO2 ((Destino3==0)&&(Destino2!=0))

// ctes del buffer de tareas
#define AVANCE               1      // tarea de avance
#define GIRO                 2  // tarea de giro
#define FINAL                0  // ultima tarea del camino
#define MAX_TAREAS    50

/*****************************************************************/
/*               Area de vbles globales EXTERNAS                 */
/*****************************************************************/
extern FILE *Mapa;

typedef struct{
       float x;
       float y;
       float Theta;
}Posicion;

typedef struct{
       float xCentro;
       float yCentro;
       float Radio;
}Giro;

typedef struct{
```

```
        Posicion Origen;
        Posicion Destino;
        Giro Circulo;
        int Status;
}Trayectoria;

extern Trayectoria BufferTareas[MAX_TAREAS];
                      // las que va preparando el generador
extern int NumTareaGenera;
```

- *controla.c*

```
/*****************************************************************/
/*            Controlador de Trayectorias                        */
/*             TFC Marta Marron. Navegacion                      */
/*                                                               */
/*             controla.c            8/8/00                      */
/*                                                               */
/*****************************************************************/
#include "controla.h"

/*****************************************************************/
/*               Area de vbles globales a este fuente            */
/*****************************************************************/
static Posicion PosicionDeseo;

/*****************************************************************/
/*               Area de vbles globales en todo el proyecto      */
/*****************************************************************/
/*****************************************************************/
/*               Area de vbles globales EXTERNAS                 */
/*****************************************************************/
// Declaradas en  controla.h
// todas las vble de posicion (deseo, real, consignas generador)
// se guardan en m ESCALADOS

//extern Posicion Posicion_Real
//extern Trayectoria BufferTareas[MAX_TAREAS];
//extern int NumTareaControl;
//extern int ObjetivoAlcanzado;

// solo para simulacion
// extern FILE *Proceso;
// extern FILE *Movil;
// extern FILE *Sigueme;
// extern FILE *Consigna;

/*****************************************************************/
/*                                    Funciones                  */
/*****************************************************************/
//---------------------------------------------------------------
// Nombre: Resta_Enteros
// Funcion:
//             Permite restar acumulacion de pulsos o ticks, evitando
//             problemas de overflow.
// Parametros entrada:
//             signed int Anterior-> anterior valor acumulado
//             signed int Siguiente-> siguiente valor acumulado
// Parametros salida:
//             signed int -> el incremento
//---------------------------------------------------------------

signed int Resta_Enteros(signed int Anterior,signed int Siguiente)
{
        signed long Incremento;
        signed int  Retorno;

        Incremento=(signed long)(Siguiente-Anterior);
        if(Anterior>Siguiente)
                        Incremento+=65536;

        Retorno=(signed int)Incremento;
```

```
        return Retorno;
}

//-----------------------------------------------------------------
// Nombre: Posicion_Deadreckoning
// Funcion:
//             Permite obtener la posicion de la silla a partir de
//             la informacion del deadreckoning (encoder) recibida por el
//     puerto paralelo.
// Parametros entrada: (de momento las consignas. Responde perfect. al modelo,
// solo para pruebas)
//     Se van a emplear: el semaforo 0 para comunicacion PC->SILLA
//     en direccion 0x0 a 0x3 (hexa) (2*2 bytes para cada integer)
//     el semaforo 1 para comunicacion SILLA->PC
//     en direccion 0x10 a 0x13 (hexa) (2*2 byes para cada integer)
// Parametros salida:
//             (No necesita pues es vble global: posicion real)
//-----------------------------------------------------------------

void Posicion_Deadreckoning(float ConsignaOmega, float ConsignaV)
{
        signed int ReciboDeadReckoning[4];
        signed int EncoDerecha,EncoIzquierda;
        unsigned int TicksDerecha,TicksIzquierda, MediaTicks;
        static signed int EncoDereAnt=0;
        static signed int EncoIzqAnt=0;
        static unsigned int TicksDereAnt=0;
        static unsigned int TicksIzqAnt=0;
        float V,Omega,x,y,WDerecha,WIzquierda,WIzqui,WDere;
        char NumeroFP[20];
        static int error=0;
        int timeout;
        signed long Aux1,Aux2;

        // lo del puerto paralelo seria:
        // recojo el semaforo 1 Silla->PC, escribo y lo suelto
        catchSemaphore(1);

        if(readSemaphore(1)!=0)
        {       timeout=clock();
                while((readSemaphore(1)!=0)&&(clock()<(timeout+1)))
                        catchSemaphore(1);    // espero 1ms a que se libere
        }

        if(readSemaphore(1)!=0)       // se ha cumplido el timeout....
        {       printf("Error de timeout en comunicacion NChip->PC, nº=
                %d\n",error);
                error++;
        }
        else
                getIntArray(0x10,ReciboDeadReckoning,4);
                                        // esta en la direccion 10 hexa

        freeSemaphore(1);

        EncoDerecha=ReciboDeadReckoning[0];
        TicksDerecha=ReciboDeadReckoning[1];
        EncoIzquierda=ReciboDeadReckoning[2];
        TicksIzquierda=ReciboDeadReckoning[3];

        // ahora se obtiene la velocidad en mrad/s... para 1º prueba solo mando 1
        EncoDerecha=Resta_Enteros(EncoDereAnt,EncoDerecha);
        EncoIzquierda=Resta_Enteros(EncoIzqAnt,EncoIzquierda);
        TicksDerecha=Resta_Enteros(TicksDereAnt,TicksDerecha);
```

```
        TicksIzquierda=Resta_Enteros(TicksIzqAnt,TicksIzquierda);

        Aux1=(signed long)EncoDerecha;
        Aux2=(signed long)TicksDerecha;
        if(Aux2==0)
                puts("Error");
        WDerecha=(float)((Aux1*KCONV)/Aux2);
        Aux1=(signed long)EncoIzquierda;
        Aux2=(signed long)TicksIzquierda;
        if(Aux2==0)
                puts("Error");
        WIzquierda=(float)((Aux1*KCONV)/Aux2);

        // Almaceno para proxima vez la lectura de encoder
        EncoDereAnt=ReciboDeadReckoning[0];
        TicksDereAnt=ReciboDeadReckoning[1];
        EncoIzqAnt=ReciboDeadReckoning[2];
        TicksIzqAnt=ReciboDeadReckoning[3];

        // se aplica la cinematica inversa.... OJO, WD Y WI en mrad/s
        V=(float)((((WDerecha+WIzquierda)/2)*RR)/1000);     // sale en m/s
        Omega=(float)((((WDerecha-WIzquierda)/DR)*RR)/1000);
                                    // rad/s-rad/s=rad/s. No paso a grados por cos.

        // luego se obtiene xReal,yReal,ThetaReal
        Omega=Omega*360/(2*PI);       // paso a grados para sumarlo.
        // rotacion de ejes para pasar a SCA
        PosicionReal.Theta=fmod(PosicionReal.Theta+Omega+360,360);
        Omega=PosicionReal.Theta*2*PI/360;          // vuelvo a rad

        MediaTicks=(TicksDerecha+TicksIzquierda)/2;
        x=(V*cos(Omega)*MediaTicks*0.82*0.001)/ESCALA;
        y=(V*sin(Omega)*MediaTicks*0.82*0.001)/ESCALA;
                                    // viene en W's si llegan en rad/s, lo escalo
        PosicionReal.x+=x;          // PosicionReal siempre lo guarda escalado.
        PosicionReal.y+=y;

        /* Fin de la funcion Posicion_Deadreckoning */
}


//---------------------------------------------------------------
// Nombre: Envia_Consignas
// Funcion:
//             Es la funcion que permite pasar las consignas de velocidad
//     obtenida del controlador a la arquitectura externa
// Parametros entrada:
//             float ConsignaOmega   -> Consigna de velocida de giro a enviar.
//             float ConsignaV       -> Consigna de velocidad lineal a enviar.
// Parametros salida:
//             No tiene pues solo actua sobre el puerto paralelo.
//---------------------------------------------------------------



void Envia_Consignas(float ConsignaOmega, float ConsignaV)
{
        int EnvioConsignas[2];
        int timeout;
        static int error=0;

        EnvioConsignas[0]=(int)(((ConsignaV+(ConsignaOmega*(DR/2)))/RR)*1000);
        EnvioConsignas[1]=(int)(((ConsignaV-(ConsignaOmega*(DR/2)))/RR)*1000);
```

```
        // recojo el semaforo 0: PC->Silla, escribo y lo suelto
        catchSemaphore(0);

        // compruebo si esta libre
        if(readSemaphore(0)!=0)
        {       timeout=clock();
                while((readSemaphore(0)!=0)&&(clock()<(timeout+1)))
                        catchSemaphore(0);    // espero 1ms a que se libere
        }

        if(readSemaphore(0)!=0)       // se ha cumplido el timeout....
        {       printf("Error de timeout en comunicacion PC->NChip, nº=
                        %d\n",error);
                error++;
        }
        else
        {       sendIntArray(0x0,EnvioConsignas,2);
                sendByte(0xA,1);              // flag indica que hay nuevas consignas
        }

        freeSemaphore(0);

        /* Fin de la funcion Envia_Consignas */
}

//-----------------------------------------------------------------
// Nombre: Posicion_Deseo_Avance
// Funcion:
//             Permite obtener la consigna de posicion en cada instante
//             si el movil esta en un tramo recto (de avance)
// Parametros entrada:
//             No necesita pues usa PosicionReal....
// Parametros salida:
//             Posicion -> estructura con la posicion pedida
//-----------------------------------------------------------------

Posicion Posicion_Deseo_Avance(void)
{
        Posicion PosicionDeseoLocal;
        float m1,m2,b1,b2;

        // evitando indeterminaciones
        if((BufferTareas[NumTareaControl].Origen.Theta==90) ||
                (BufferTareas[NumTareaControl].Origen.Theta==270))
        {       PosicionDeseoLocal.x=BufferTareas[NumTareaControl].Origen.x;
                PosicionDeseoLocal.y=PosicionReal.y;
        }
        else if((BufferTareas[NumTareaControl].Origen.Theta==0) ||
                (BufferTareas[NumTareaControl].Origen.Theta==180))
        {       PosicionDeseoLocal.y=BufferTareas[NumTareaControl].Origen.y;
                PosicionDeseoLocal.x=PosicionReal.x;
        }

        // por el metodo convencional....
        else
        {       m1=tan(BufferTareas[NumTareaControl].Origen.Theta*PI/180);
                m2=-1/m1;   // la normal

                b1=BufferTareas[NumTareaControl].Origen.y-
                        m1*BufferTareas[NumTareaControl].Origen.x;
                b2=PosicionReal.y-m2*PosicionReal.x;

                PosicionDeseoLocal.x=(b1-b2)/(m2-m1);
                PosicionDeseoLocal.y=m1*PosicionDeseoLocal.x+b1;
```

```
	}

	// la orientacion es la de la consigna, es liena recta
	PosicionDeseoLocal.Theta=BufferTareas[NumTareaControl].Origen.Theta;

	return PosicionDeseoLocal;

	/* Fin de la funcion Posicion_Deseo_Avance */
}

//---------------------------------------------------------------
// Nombre: Posicion_Deseo_Giro
// Funcion:
//		Permite obtener la consigna de posicion en cada instante
//		si el movil esta en un tramo recto (de avance)
// Parametros entrada:
//		No necesita, PosicionReal
// Parametros salida:
//		Posicion -> estructura con la posicion pedida
//---------------------------------------------------------------

Posicion Posicion_Deseo_Giro(void)
{
	Posicion PosicionDeseoLocal;
	float xCentro,yCentro,R,MinThetaTramo,MaxThetaTramo;
	float Aux,Aux2,y1,y2,x1,x2,m1,m2;

	// 1º recojo parametros de giro
	xCentro=BufferTareas[NumTareaControl].Circulo.xCentro;
	yCentro=BufferTareas[NumTareaControl].Circulo.yCentro;
	R=BufferTareas[NumTareaControl].Circulo.Radio;

	// 2º obtengo el punto a traves de la ec. de la circunferencia y
	// la recta que une el punto real y el origen

	// he de eliminar la indeterminacion... no es un  buen metodo
	if(PosicionReal.y==yCentro)
	{  PosicionDeseoLocal.y=PosicionReal.y;
		PosicionDeseoLocal.x=xCentro+R;
			// modificacion, es mas logico, pero quitar si da problemas
	}

	// metodod convencional
	else
	{	Aux=(pow((PosicionReal.x-xCentro),2)/pow((PosicionReal.y-
				yCentro),2))+1;
		Aux=sqrt(Aux);
		y1=R/Aux+yCentro;
		y2=yCentro-R/Aux;

		x1=((PosicionReal.x-xCentro)/(PosicionReal.y-yCentro))*(y1-
				yCentro)+xCentro;
		x2=((PosicionReal.x-xCentro)/(PosicionReal.y-yCentro))*(y2-
				yCentro)+xCentro;

		// de las 2 soluciones elijo la que esta mas proxima al punto real
		Aux=pow((PosicionReal.x-x1),2)+pow((PosicionReal.y-y1),2);
		Aux2=pow((PosicionReal.x-x2),2)+pow((PosicionReal.y-y2),2);

		if(Aux<Aux2)
		{	PosicionDeseoLocal.x=x1;
			PosicionDeseoLocal.y=y1;
		}
		else
```

```
        {       PosicionDeseoLocal.x=x2;
                PosicionDeseoLocal.y=y2;
        }
}

// 3º despues he de hallar el angulo deseado. Se obtiene facilmetne a
// traves de la pendiente de la recta qeu contenia al punto deseado,
// entre el centro de la circunferencia y el punto real.

// la consigna de orientacion ha de esta siempre en el antorno de la del
// nodo  de entrada y la del de salida
MaxThetaTramo=max(BufferTareas[NumTareaControl].Destino.Theta,
        BufferTareas[NumTareaControl].Origen.Theta);
MinThetaTramo=min(BufferTareas[NumTareaControl].Destino.Theta,
        BufferTareas[NumTareaControl].Origen.Theta);

// amplio el rango para los entornos de los puntos de origen y fin del
// trayecto estoy fuera de la zona de discontinuidad
if((MaxThetaTramo-MinThetaTramo)<180)
{       // el 2 es un entorno de error
        MaxThetaTramo=min(MaxThetaTramo+2,360);

        if (MinThetaTramo!=0)
                MinThetaTramo=max(MinThetaTramo-2,0);

        // si es exactamente 0, pasamos de no estar en zona de
        // discontinuidad a si estarlo
        else
        {       MinThetaTramo=MaxThetaTramo;
                MaxThetaTramo=358;
        }
}

// estoy dentro de la zona de discontinuidad
else
{       MaxThetaTramo=MaxThetaTramo-2;
        MinThetaTramo=MinThetaTramo+2;
}

// Ahora calculo la consigna de orientacion
// 1º elimino indeterminaciones..... por poner algo
if(PosicionReal.x==xCentro)
        PosicionDeseoLocal.Theta=0;

else if(PosicionReal.y==yCentro)
        PosicionDeseoLocal.Theta=90;

// por el metodo convencional
else
{       m1=(yCentro-PosicionReal.y)/(xCentro-PosicionReal.x);
        m2=atan(-1/m1);
        m2=m2*360/(2*PI);
        PosicionDeseoLocal.Theta=fmod(m2+360,360);
                                                        // por si sale negativo
}

// Ahora me aseguro de que la orientacion este entre el min y el max
// fuera de la zona de discontinuidad
if((MaxThetaTramo-MinThetaTramo)<180)
{       if((PosicionDeseoLocal.Theta<MinThetaTramo)||
                        (PosicionDeseoLocal.Theta>MaxThetaTramo))
                PosicionDeseoLocal.Theta=fmod(PosicionDeseoLocal.Theta+180,360);
}
```

```
        // estoy ne zona de discontinuidad, la condicion es la opuesta
        else
        {       if((PosicionDeseoLocal.Theta>MinThetaTramo)&&
                                (PosicionDeseoLocal.Theta<MaxThetaTramo))
                        PosicionDeseoLocal.Theta=fmod(PosicionDeseoLocal.Theta+180,360);
        }

        PosicionDeseoLocal.Theta=fmod(PosicionDeseoLocal.Theta,360);
                                                                // por si sale > 360
        return PosicionDeseoLocal;

        /* Fin de la funcion Posicion_Deseo_Giro */
}

//-----------------------------------------------------------------
// Nombre: Controla_Omega
// Funcion:
//             Es la funcion a la que se llama periodicamente para generar la
//             consigna de Omega a partir de la info. proporcionada por el
//             generador de   trayectorias y el deadreckoning.
// Parametros entrada:
//             La info que necesita se encuentra en el array de estructuras
//             global
//             BufferTareas de tipo Trayectoria.
//             Lo mismo ocurre con las funciones Posicion_Deseo_Avance/Giro
// Parametros salida:
//             float -> velocidad angular a aplicar.
//-----------------------------------------------------------------

float Controla_Omega()
{
        float ErrorDesplazamiento, ErrorOrientacion;
        float ConsignaOmega;

        //----------------------------------------
        // 1º se obtiene el error.
        //----------------------------------------

        ErrorDesplazamiento=pow((PosicionDeseo.x-PosicionReal.x),2)+
                                        pow((PosicionDeseo.y-PosicionReal.y),2);
        ErrorDesplazamiento=sqrt(ErrorDesplazamiento);
        ErrorOrientacion=PosicionDeseo.Theta-PosicionReal.Theta;

        //hasta aqui el error esta entre +90 y -90 (sino hubiese ido a
        // Movil_Desorientado
        // pero por si acaso tengo en cuenta el angulo >180
        ErrorOrientacion=fmod(ErrorOrientacion,360);// por si mayor
        if(ErrorOrientacion>180)     ErrorOrientacion-=360;
                                                                // para hacerlo negativo
        else if (ErrorOrientacion<-180) ErrorOrientacion+=360;

        //----------------------------------------
        // 2º se aplica el controlador.
        //----------------------------------------

        // La cte es distinta en funcion de si es avance o giro
        if(BufferTareas[NumTareaControl].Status==AVANCE)
                ConsignaOmega=ErrorDesplazamiento*KD_AVANCE+
                                KO_AVANCEfabs(ErrorOrientacion);

        else if(BufferTareas[NumTareaControl].Status==GIRO)
                ConsignaOmega=ErrorDesplazamiento*KD_GIRO+KO_GIRO*fabs(ErrorOrientacion);

        // para evitar efecto de valor siempre positivo de ErrorDesplazamiento
```

```
        // le doy el signo de ErrorOrientacion a la consigna, que es +
        if((ErrorOrientacion<0)&&(ConsignaOmega>0))
                ConsignaOmega=(-1)*ConsignaOmega;

        //  hago correccion para que los angulos sean entre +-180
        if(ConsignaOmega>180)
                ConsignaOmega-=360;
        else if (ConsignaOmega<-180)
                ConsignaOmega+=360;

        // saturando....      TENDRIA QUE HACERLO DESPUES DE FILTRO ACELERACION
        if(ConsignaOmega>OMEGA_MAX)
                ConsignaOmega=OMEGA_MAX;
        else if (ConsignaOmega<((-1)*OMEGA_MAX))
                ConsignaOmega=(-1)*OMEGA_MAX;

        // y paso a rad/s
        ConsignaOmega=ConsignaOmega*(2*PI)/360;

        return ConsignaOmega;
        // sale en rad/s

        /* Fin de la funcion Controla_Omega */
}

//-----------------------------------------------------------------
// Nombre: Controla_V
// Funcion:
//             Es la funcion a la que se llama periodicamente para generar la
//             consigna de Velociadad Lineal a traves de la info del generador de
//             trayectorias y el deadreckoning.
// Parametros entrada:
//float DistanciaDeceleracion -> Distancia que permite diseñar el perfil de V
//float VGiro  -> Velocidad de giro en tarea de avance actual o proxima
// Parametros salida:
//             float -> velocidad lineal a aplicar.
//-----------------------------------------------------------------

float Controla_V(float DistanciaDeceleracion,float VGiro)
{
        float ConsignaV, DistanciaDestino;

        // si es tarea de avance
        if(BufferTareas[NumTareaControl].Status==AVANCE)
        {  // Calculo la distancia Euclidea hasta el final
                DistanciaDestino=pow((BufferTareas[NumTareaControl].Destino.x-
                PosicionReal.x),2)+pow((BufferTareas[NumTareaControl].Destino.y-
                PosicionReal.y),2);
                DistanciaDestino=sqrt(DistanciaDestino);

                // comprobando si hay qeu decelerar
                if(DistanciaDestino<=DistanciaDeceleracion)
                        ConsignaV=V_MAX;      // en m/s
                else ConsignaV=VGiro;
        }

        // si es tarea de giro
        else if(BufferTareas[NumTareaControl].Status==GIRO)
                ConsignaV=VGiro;

        // sale en m/s
        return ConsignaV;
        /* Fin de la funcion Controla_V */
}
```

```
//---------------------------------------------------------------
// Nombre: Movil_Desorientado
// Funcion:
//            Es la funcion a la que se llama como controlador de Omega, cuando
//            el movil esta avanzando en sentido contrario al que deberia
// Parametros entrada:
//     La info que necesita  se encuentra en el array de estructuras global
//            BufferTareas de tipo Trayectoria
// Parametros salida:
//            float -> velocidad angular a aplicar.
//---------------------------------------------------------------

float Movil_Desorientado(void)
{
       float ConsignaOmega, ConsignaOrienta, ErrorOrienta;

       // 1º se obtiene la consigna de orientacion que deseo (no va a ser
       //  PosicionDeseo.Theta)
       if((PosicionDeseo.x-PosicionReal.x)==0)
       {      if((PosicionDeseo.y-PosicionReal.y)>0)      ConsignaOrienta=90;
              else if((PosicionDeseo.y-PosicionReal.y)<0) ConsignaOrienta=270;
              else ConsignaOrienta=PosicionDeseo.Theta;
       }

       else
       {      ConsignaOrienta=atan((PosicionDeseo.y-
              PosicionReal.y)/(PosicionDeseo.x-PosicionReal.x));
              ConsignaOrienta=ConsignaOrienta*360/(2*PI);
              if(ConsignaOrienta<0) ConsignaOrienta+=360;
              ConsignaOrienta=fmod(ConsignaOrienta,360);
       }

// 2º Ahora se comprueba que no estemos en el 2º o 3º cuadrante. Hay que sumar 180
       if((PosicionDeseo.x-PosicionReal.x)<0)
       {      ConsignaOrienta+=180;
              ConsignaOrienta=fmod(ConsignaOrienta,360);
       }

       // 3º Se obtiene el error de consigna y se gira de forma proporcional a este
       ErrorOrienta=ConsignaOrienta-PosicionReal.Theta;
       ConsignaOmega=ErrorOrienta*KO_DESORIENTO*OMEGA_MAX;

       if(ConsignaOmega>OMEGA_MAX) ConsignaOmega=OMEGA_MAX;
       else if (ConsignaOmega<((-1)*OMEGA_MAX)) ConsignaOmega=(-1)*OMEGA_MAX;

       // y paso a rad/s
       ConsignaOmega=ConsignaOmega*(2*PI)/360;

       return ConsignaOmega;

       /* Fin de la funcion Movil_Desorientado */

}

//---------------------------------------------------------------
// Nombre: Filtro_Aceleracion
// Funcion:
//            Es la funcion a la que se llama para evitar cambios bruscos
//            de velocidad.
//            En ppio solo se filtra la velocidad lineal.
// Parametros entrada:
//            float *VeloGiro   -> la velocidad de giro, omega
//            float *VeloAvance -> la velocidad de avance, V
```

```
// Parametros salida:
//             No habra paramertos de salida, parametros pasados por referencia
//-----------------------------------------------------------------

void Filtro_Aceleracion(float *VeloGiro, float *VeloAvance)
{
        static float VeloAvanceAnterior=0;
        static float VeloGiroAnterior=0;

        // con velocidad de avance
        if((*VeloAvance-VeloAvanceAnterior)>V_MAX_ACELERA)
                *VeloAvance=VeloAvanceAnterior+V_MAX_ACELERA;

        else if((*VeloAvance-VeloAvanceAnterior)<((-1)*V_MAX_ACELERA))
                *VeloAvance=VeloAvanceAnterior-V_MAX_ACELERA;

         VeloAvanceAnterior=*VeloAvance;

        // con velocidad de  giro
        if((*VeloGiro-VeloGiroAnterior)>OMEGA_MAX_ACELERA)
                *VeloGiro=VeloGiroAnterior+OMEGA_MAX_ACELERA;

        else if((*VeloGiro-VeloGiroAnterior)<((-1)*OMEGA_MAX_ACELERA))
                *VeloGiro=VeloGiroAnterior-OMEGA_MAX_ACELERA;

         VeloGiroAnterior=*VeloGiro;

        /* Fin de la funcion Filtro_Aceleracion*/
}

//-----------------------------------------------------------------
// Nombre: Controla
// Funcion:
//             Es la funcion a la que se llama periodicamente para generar las
//             consignas a aplicar a los motores a partir de la inforamcion
//             del generador de trayectorias y el deadreckoning.
// Parametros entrada:
//     no tiene, puesto que nunca se le llama. La info que necesita
//     se encuentra en el array de estructuras BufferTareas de tipo Trayectoria
// Parametros salida:
//             Tampoco tiene.
//-----------------------------------------------------------------

void Controla(void)
{
        static float DistanciaDeceleracion;
        static float VGiro,VGiroAnterior;
        static float MaxTheta,MinTheta;
        static int TareaLista=0;

        // es static poruqe para las pruebas obtengo la lectura de encoders a
        // partir de ellas
        static float ConsignaOmega=0;
        static float ConsignaV=0;

        // auxiliares
        float Distancia;
        char NumeroFP[20];


        //-----------------------------------------
        // 1º se obtiene la posicion real
        //-----------------------------------------
        // es importante que se haga la acumulacion de pulsos, independientemente
```

```
// de que haya o no trayectoria.
// de momento se hace que responda perfectamente al modelo, asi que envio
// las consignas....

Posicion_Deadreckoning(ConsignaOmega, ConsignaV);

//-----------------------------------------
// 2º se comprueba que no he acabado toda la trayectoria
//-----------------------------------------

if(TareaLista==1)
{
	//-----------------------------------------
	// 3º se obtiene la posicion deseada (consigna en este instante)
	//-----------------------------------------

	if(BufferTareas[NumTareaControl].Status==AVANCE)
PosicionDeseo=Posicion_Deseo_Avance();
	else 			if(BufferTareas[NumTareaControl].Status==GIRO)
PosicionDeseo=Posicion_Deseo_Giro();


	//-----------------------------------------
	// 4º se comprueba si el movil esta totalmente desorientado
	//-----------------------------------------

	// se considera que es asi cuando no esta entorno a +-90º
	MaxTheta=fmod(PosicionDeseo.Theta+90,360);
	MinTheta=fmod(PosicionDeseo.Theta+270,360);

	//-----------------------------------------
	// 5º en funcion de eso se llama a uno u otro controlador.
	//-----------------------------------------

	// el entorno incluye la zona de discontinuidad
	if(MinTheta>MaxTheta)
	{	// compruebo que esta en fuera del rango +-90º
		if((PosicionReal.Theta>MinTheta)&&(PosicionReal.Theta<MaxTheta))
		{	ConsignaOmega=Movil_Desorientado();
			ConsignaV=0;   // y lo paro
		}

		// sino es que esta dentro.
		else
		{	ConsignaOmega=Controla_Omega();
			ConsignaV=Controla_V(DistanciaDeceleracion,VGiro);
		}
	}

	// el entorno no incluye la zona de discontinuidad.
	else
	{	// esta fuera del rango +-90º
		if((PosicionReal.Theta<MinTheta)||(PosicionReal.Theta>MaxTheta))
		{	ConsignaOmega=Movil_Desorientado();
			ConsignaV=0;   // y lo paro
		}

		// sino esta dentro.
		else
		{	ConsignaOmega=Controla_Omega();
			ConsignaV=Controla_V(DistanciaDeceleracion,VGiro);
		}
	}
```

```
        //----------------------------------------
        // 6º paso el filtro de aceleracion y se envian las nuevas
        // consignas
        //----------------------------------------

        Filtro_Aceleracion(&ConsignaOmega,&ConsignaV);
        Envia_Consignas(ConsignaOmega,ConsignaV);

        //----------------------------------------
        // 7º se comprueba que no se ha alcanzado el destino de la tarea
        // actual
        //----------------------------------------
        Distancia=pow((BufferTareas[NumTareaControl].Destino.x-
        PosicionReal.x),2)+pow((BufferTareas[NumTareaControl].Destino.y-
        PosicionReal.y),2);

        if(Distancia<= ENTORNO)
        {       printf("tarea nº %i completada\n",NumTareaControl-1);
                TareaLista=0;

// Ademas se comprueba si se ha completado un tramo entre dos nodos

                // se acabo el recorrido
                if(BufferTareas[NumTareaControl+1].Status==FINAL)
                {       puts("Final del recorrido: DESTINO ALCANZADO!!!");
                        ConsignaV=0;
                        ConsignaOmega=0;
                        BufferTareas[NumTareaControl+1].Status=-1;
                        BufferTareas[NumTareaControl].Status=-1;
                        ObjetivoAlcanzado=1;
                                        // finaliza el proceso de navegacion.
                }

                // esperando que se llene el buffer de nuevo
                else if((BufferTareas[NumTareaControl+2].Status
                ==-1)||(BufferTareas[NumTareaControl+1].Status==-1))
                {  puts("Esperando se llene el buffer tareas...");
                        ConsignaV=0;
                        ConsignaOmega=0;
                }

                // sino es uqe hay una tarea nueva
                else  if((BufferTareas[NumTareaControl+2].Status!=-1)&&
                        (BufferTareas[NumTareaControl+1].Status!=-1))
                {  puts("Afrontando nueva tarea");
                        BufferTareas[NumTareaControl].Status=-1;
                        NumTareaControl=(NumTareaControl+1)%MAX_TAREAS;
                        TareaLista=1;

                // actualizando los parametros de velocidad en funcion de
                // las tareas anterior y siguiente
                        if(BufferTareas[NumTareaControl].Status==AVANCE)
                {  // obteninendo vgiro de tarea anterior y de siguietne
                                VGiroAnterior=VGiro;

VGiro=BufferTareas[NumTareaControl+1].Circulo.Radio*KV_GIRO;

                                // Obtenidno distancia de desaceleracion

Distancia=pow((BufferTareas[NumTareaControl].Origen.x-
BufferTareas[NumTareaControl].Destino.x),2)+pow((BufferTareas
[NumTareaControl].Destino.y-BufferTareas[NumTareaControl].Origen.y),2);
                                Distancia=sqrt(Distancia);
```

```
        DistanciaDeceleracion=(pow(VGiroAnterior,2)/CTE_V_ACELERA)-
                                (pow(VGiro,2)/CTE_V_ACELERA);
                                        DistanciaDeceleracion+=Distancia/2;
                                }

                                else if(BufferTareas[NumTareaControl].Status==GIRO)
                                VGiro=BufferTareas[NumTareaControl].Circulo.Radio*KV_GIRO;
                        }

                // del fin de tarea
                }

        // de la tarea lista
        }

        //----------------------------------------
        // 8º Si no hay tarea entonces se para el movil
        //----------------------------------------

        else
        {  // se comprueba si ya sehan llenado mas pos. en los buffers
                if((BufferTareas[NumTareaControl+1].Status!=-1)
                        &&BufferTareas[NumTareaControl+2].Status!=-1))
                {       TareaLista=1;
                        BufferTareas[NumTareaControl].Status=-1;
                        NumTareaControl=(NumTareaControl+1)%MAX_TAREAS;
                }
        }

        /* Fin de la funcion Controla */
}
```

- *controla.h*

```
/*****************************************************************/
/*       Controlador de Trayectorias. Fichero cabecera           */
/*             TFC Marta Marron. Navegacion                      */
/*                                                               */
/*     controla.h            8/8/00                              */
/*****************************************************************/

/*****************************************************************/
/*                    Area de librerias                          */
/*****************************************************************/
#include <string.h>
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>


/*****************************************************************/
/*                          Area de constantes                   */
/*****************************************************************/
// ctes para cambios de espacio de dimension
#define RR     0.16           // Es el radio de la rueda de la silla 16cm
#define DR     0.52           // Es la distancia entre ruedas 1/2m=52cm
#define PI     3.1415926
#define ESCALA 5      // Para obtener x,y en la escala del mapa. (cada cm son 5m)
#define KCONV  598.628// para convertir a mrad/s: mrad/s=(pulsos*KCONV)/Ts(en ticks)

// ctes para el controlador de omega
#define KD_AVANCE     1.3*90         // (15*90) Cte error desplazamiento. Si error
                                     // 1/2cm escalados=2.5m-> 90º, 0.36º/cm
#define KO_AVANCE     0.01*90// (0.001*90) Cte error orientacion. Si error
                                     // de 1/0.007º=142 ->90º, 0.63º/º
#define KD_GIRO       90             // (15*90) Cte error desplazamiento. Si error
                                     // 1/1cm escalados=5m-> 90º, 0.18º/cm
#define KO_GIRO       0.007*90       // (0.001*90) Cte error orientacion. Si error
                                     // de 1/0.007º=142 ->90º, 0.63º/º
#define KO_DESORIENTO 0.011   // Cte error orientacion si esta perdido. Si es
                                     // de 90 grados la saldia es maxima
#define OMEGA_MAX     150            // en grados/s tal y como esta puesto la
                      // obtencion de la consigna la Omega maxima es el doble.

// ctes para el controlador de V
#define KV_GIRO                      1
#define V_MAX                        0.7     // en m/s    eran 0.3

// ctes para el filtro de aceleracion
#define V_MAX_ACELERA         0.02    // en m/s2     era 0.02
#define OMEGA_MAX_ACELERA     0.07    // en rad/s   era 0.1
#define CTE_V_ACELERA         V_MAX_ACELERA*TS*4

// ctes para el buffer circular
#define MAX_TAREAS    30// Mx nº de tareas que puede tener la trayectoria total
#define AVANCE        1       // Estado indica que la tarea es de avance.
#define GIRO          2       // Estado que indica que la tarea es de giro.
#define FINAL         0       // Estado que indica que ya no hay mas tareas en el
                              // camino completo.

// cte para determinar destino alcanzado
#define ENTORNO       0.0004  // Esta elevada al cuadrado .Un radio de 10.cm
#define TS            50                    // el periodo de muestreo
```

```
/*****************************************************************/
/*           Area de vbles globales EXTERNAS                     */
/*****************************************************************/
typedef struct{
        float x;
        float y;
        float Theta;
}Posicion;

typedef struct{
        float xCentro;
        float yCentro;
        float Radio;
}Giro;

typedef struct{
        Posicion Origen;
        Posicion Destino;
        Giro Circulo;
        int Status;
}Trayectoria;

extern Posicion PosicionReal;
extern Trayectoria BufferTareas[MAX_TAREAS];
extern int NumTareaControl;
extern int ObjetivoAlcanzado;

// solo para simulacion
extern FILE *Proceso;
extern FILE *Sigueme;
extern FILE *Consigna;
extern FILE *Movil;
```

- *planruta.c*

```
/*****************************************************************/
/*            Planificador de rutas.                             */
/*     TFC Marta Marron. Navegacion                              */
/*                                                               */
/*     planruta.c            8/4/00                               */
/*****************************************************************/
#include "planruta.h"

/*****************************************************************/
/*           Area de vbles globales de todo el proyecto          */
/*****************************************************************/

/*****************************************************************/
/*              Area de vbles globales en este fuente            */
/*****************************************************************/
static int Origen3,Origen2,Origen1,Destino3,Destino2,Destino1;
static int i=0;
static int NodoOrigen, NodoDestino;
static int ListaNodos[MAX_NODOS_MAPA];

/*****************************************************************/
/*              Area de vbles externas                           */
/*****************************************************************/
// Definidas en planruta.h
// extern FILE *Mapa;
// extern Ruta[MAX_NODOS_MAPA];

/*****************************************************************/
/*                           Funciones                           */
/*****************************************************************/

/*---------------------------------------------------------------*/
// Nombre: Introduce_Transicion
// Funcion:
//     Permite anadir a la lista de nodos de la ruta los nodos de
//             transicion que sean necesarios para el generador.
// Parametros entrada:
// Parametros salida:
//
/*---------------------------------------------------------------*/

void Introduce_Transicion(void)
{
       int i=0;
       int j=0;
       int Nodo1,Nodo2;
       int Planta1,Planta2,Pasillo1, Pasillo2, Sala1, Sala2;

       while(ListaNodos[i]!=NodoDestino)
       {  /* Incorporar el 1º de los nodos, veremos si meter transicion */
              Ruta[j++]=ListaNodos[i];

              /* Obtengo los nodos y la planta y pasillo donde se encuentran */
              Nodo1=ListaNodos[i++];
              Nodo2=ListaNodos[i];
              Planta1=Nodo1/100;
              Planta2=Nodo2/100;
              Pasillo1=(Nodo1%100)/10;
              Pasillo2=(Nodo2%100)/10;
              Sala1=Nodo1%10;
              Sala2=Nodo2%10;

              /* Solo hay transicion entre nodos de la misma planta. */
              if(Planta1==Planta2)
              {
                     /* entonces chequeo estoy nivel 2 jerarq (salas comunes)*/
                     if((Pasillo1==0) && (Pasillo2==0))
                     {      /* Ahora chequeo en funcion de la planta */
                            switch(Planta1)       /* ... o planta 2.... */
                            {  case 0,1:
       if((Sala1<=SALA_SUBRED1_01) && (Sala2>SALA_SUBRED1_01))
                                          Ruta[j++]=(-1)*(Planta1*100+1);
       else if((Sala1>SALA_SUBRED1_01) && (Sala2<=SALA_SUBRED1_01))
                                          Ruta[j++]=(-1)*(Planta1*100+1);
```

```
                                        break;
                        case 2:
if((Sala1<=SALA_SUBRED1_2) && (Sala2>SALA_SUBRED1_2))
                                        Ruta[j++]=(-1)*(Planta1*100+1);
else if((Sala1>SALA_SUBRED1_2) && (Sala2<=SALA_SUBRED1_2))
                                Ruta[j++]=(-1)*(Planta1*100+1);
                                        break;
                        case 3:
if((Sala1<=SALA_SUBRED1_3) && (Sala2>SALA_SUBRED1_3))
                                        Ruta[j++]=(-1)*(Planta1*100+1);
if((Sala1<=SALA_SUBRED2_3) && (Sala2>SALA_SUBRED2_3))
                                        Ruta[j++]=(-1)*(Planta1*100+2);
else if((Sala1>SALA_SUBRED2_3) && (Sala2<=SALA_SUBRED2_3))
                                Ruta[j++]=(-1)*(Planta1*100+2);
if((Sala1>SALA_SUBRED2_3) && (Sala2<=SALA_SUBRED1_3))
                                        Ruta[j++]=(-1)*(Planta1*100+1);
else if((Sala1>SALA_SUBRED1_3) && (Sala1<=SALA_SUBRED2_3) &&
(Sala2<=SALA_SUBRED1_3))
                                        Ruta[j++]=(-1)*(Planta1*100+1);
else if((Sala1>SALA_SUBRED2_3) && (Sala1<=SALA_SUBRED2_3) &&
(Sala2>SALA_SUBRED2_3))
                                        Ruta[j++]=(-1)*(Planta1*100+1);
                                        break;
                        }
                }

                /*ahora chequeo estoy salas comunes nodos jerarq nivel2  */
                else if (Pasillo1==0)
                        switch (Planta1)
                        {  case 0,1:
        if((Sala1<=SALA_SUBRED1_01) && (Pasillo2>PASO_SUBRED1_01))
                                        Ruta[j++]=(-1)*(Planta1*100+1);
        else if((Sala1>SALA_SUBRED1_01) && Pasillo2<=PASO_SUBRED1_01))
                                        Ruta[j++]=(-1)*(Planta1*100+1);
                                        break;
                        case 2:
        if((Sala1<=SALA_SUBRED1_2) && (Pasillo2>PASO_SUBRED1_2))
                                        Ruta[j++]=(-1)*(Planta1*100+1);
        else if((Sala1>SALA_SUBRED1_2) && (Pasillo2<=PASO_SUBRED1_2))
                                        Ruta[j++]=(-1)*(Planta1*100+1);
                                        break;
                        case 3:
        if((Sala1<=SALA_SUBRED1_3) && (Pasillo2>PASO_SUBRED1_3))
                        {               Ruta[j++]=(-1)*(Planta1*100+1);
                                        Ruta[j++]=(-1)*(Planta1*100+2);
                        }
        else if((Sala1>SALA_SUBRED2_3) && (Pasillo2<=PASO_SUBRED1_3))
                        {               Ruta[j++]=(-1)*(Planta1*100+2);
                                        Ruta[j++]=(-1)*(Planta1*100+1);
                        }
        else if((Sala1<=SALA_SUBRED2_3) && (Sala1>SALA_SUBRED1_3) &&
        (Pasillo2>PASO_SUBRED1_3))
                                        Ruta[j++]=(-1)*(Planta1*100+2);
        else if((Sala1<=SALA_SUBRED2_3) && (Sala1>SALA_SUBRED1_3) &&
        (Pasillo2<=PASO_SUBRED1_3))
                                        Ruta[j++]=(-1)*(Planta1*100+1);
                                        break;
                        }

                /*Finalmente ver si paso nivel jerarquico a sala comun */
                else if (Pasillo2==0)
                        switch (Planta1)
                        {  case 0,1:
        if((Sala2<=SALA_SUBRED1_01) && (Pasillo1>PASO_SUBRED1_01))
                                Ruta[j++]=(-1)*(Planta1*100+1);
        else if((Sala2>SALA_SUBRED1_01) && (Pasillo1<=PASO_SUBRED1_01))
                                        Ruta[j++]=(-1)*(Planta1*100+1);
                                        break;
                        case 2:
        if((Sala2<=SALA_SUBRED1_01) && (Pasillo1>PASO_SUBRED1_2))
                                        Ruta[j++]=(-1)*(Planta1*100+1);
        else if((Sala2>SALA_SUBRED1_01) && (Pasillo1<=PASO_SUBRED1_2))
                                        Ruta[j++]=(-1)*(Planta1*100+1);
                                        break;
                        case 3:
        if((Sala2<=SALA_SUBRED1_3) && (Pasillo1>PASO_SUBRED1_3))
                        {               Ruta[j++]=(-1)*(Planta1*100+2);
```

```
                                        Ruta[j++]=(-1)*(Planta1*100+1);
                                }
                else if((Sala2>SALA_SUBRED2_3) && (Pasillo1<=PASO_SUBRED1_3))
                                {               Ruta[j++]=(-1)*(Planta1*100+1);
                                                Ruta[j++]=(-1)*(Planta1*100+2);
                                }
                else if((Sala2<=SALA_SUBRED2_3) && (Sala2>SALA_SUBRED1_3) &&
                (Pasillo1>PASO_SUBRED1_3))
                                                Ruta[j++]=(-1)*(Planta1*100+2);
                else if((Sala2<=SALA_SUBRED2_3) && (Sala2>SALA_SUBRED1_3) &&
                (Pasillo1<=PASO_SUBRED1_3))
                                                Ruta[j++]=(-1)*(Planta1*100+1);
                                                break;
                                }

                /* Queda por unir dos nodos jerarquicos entre si */
                else if((Sala1==0) && (Sala2==0))
                        switch(Planta1)
                        {       case 0,1:
                                if((Pasillo1!=CENTRAL_01)&& (Pasillo2!=CENTRAL_01))
        {       if((Pasillo1<=PASO_SUBRED1_01) && (Pasillo2>PASO_SUBRED1_01))
                                                Ruta[j++]=(-1)*(Planta1*100+1);
                if((Pasillo1>PASO_SUBRED1_01) && (Pasillo2<=PASO_SUBRED1_01))
                                                Ruta[j++]=(-1)*(Planta1*100+1);
        }
                        break;
                                case 2:
                                if((Pasillo1!=CENTRAL_2)&& (Pasillo2!=CENTRAL_2))
         {      if((Pasillo1<=PASO_SUBRED1_2) && (Pasillo2>PASO_SUBRED1_2))
                                                Ruta[j++]=(-1)*(Planta1*100+1);
                if((Pasillo1>PASO_SUBRED1_2) && (Pasillo2<=PASO_SUBRED1_2))
                                                Ruta[j++]=(-1)*(Planta1*100+1);
         }
                                break;
                                case 3:
                if((Pasillo1<=PASO_SUBRED1_3) && (Pasillo2>PASO_SUBRED1_3))
                                {       Ruta[j++]=(-1)*(Planta1*100+1);
                                                Ruta[j++]=(-1)*(Planta1*100+2);
                                }
                if((Pasillo1>PASO_SUBRED1_3) && (Pasillo2<=PASO_SUBRED1_3))
                                        {       Ruta[j++]=(-1)*(Planta1*100+2);
                                                Ruta[j++]=(-1)*(Planta1*100+1);
                                        }
                                break;
                                }
        /* Solventando el problema de nodos de transicion en pasillo central */
        if((Pasillo1==CENTRAL_01) && (Pasillo2==CENTRAL_01) && (Planta1==1))
        {       if((Sala1==INTERIOR1_12) || (Sala1==INTERIOR2_12))
                                        Ruta[j++]=(-1)*(Planta1*100+30+1);
                        else if((Sala2==INTERIOR1_12) || (Sala2==INTERIOR2_12))
                                        Ruta[j++]=(-1)*(Planta1*100+30+1);
                                }
                if((Pasillo1==CENTRAL_2)&& (Pasillo2==CENTRAL_2) && (Planta1==2))
{  if((Sala1==INTERIOR1_12)||
                (Sala1==INTERIOR2_12))Ruta[j++]=(-1)*(Planta1*100+50+1);
else if((Sala2==INTERIOR1_12)|| (Sala2==INTERIOR2_12))Ruta[j++]=(-1)*(Planta1*100+50+1);
}

}
}

Ruta[j]=NodoDestino;

/* Fin de funcion Introduce_Transicion*/

}

/*---------------------------------------------------------------*/
// Nombre: Ascensor_Proximo
// Funcion:
//     Permite anadir a la lista de nodos de la ruta el camino al
//             ascensor mas proximo del origen dado
// Parametros entrada:
//             int NodoPartida -> nombre del nodo de partida
// Parametros salida:
//
/*---------------------------------------------------------------*/
```

```
void Ascensor_Proximo(int NodoPartida)
/* esta funcion mete en la lista de nodos la ruta necesaria hasta
/* alcanzar el ascensor mas proximo al nodo pasado */
{
        char *Nodo;
        int Ascensor;
        float Distancia1,Distancia2,Distancia5,Distancia6
        float xPartida,yPartida,x1,y1,x2,y2,x5,y5,x6,y6;

        printf("Buscando Ascensor proximo al nodo %d\n",NodoPartida);
        /* Busqueda de la posicion x,y del NodoPartida */
        Nodo=(char *)Buscar_Nodo(NodoPartida);

        /* En "nodo" tengo el nodo cuyo nombre coincide con NodoPartida */
        /* Ahora recoger el campo 2º y 3º son respectivamente posicion x e y */
        xPartida=atof(strtok('\0'," ")); /* El segundo campo es x */
        yPartida=atof(strtok('\0'," ")); /* El tercer campo es y */

        /* hay que hallar cual es la distancia a cada uno de los ascensores */
        /* Existen 4 ascensores por planta, cuyo nombr es siempre: 1ºjerarquia+0+1,2,5,6
        */
        /* En la planta 3º solo existen el 1 y el 6 */

        /* Busco el ascensor 1ºjerarquia+0+1 */
        Ascensor=(int)(NodoPartida/100)*100+1;
        Nodo=(char *)Buscar_Nodo(Ascensor);

        /* En la variable "nodo" tengo el nodo cuyo nombre coincide con ascensor1*/
        /* Ahora hay que recoger el campo 2º y 3º que son respectivamente la posicion x e
        y */
        x1=atof(strtok('\0'," "));       /* El segundo campo es x */
        y1=atof(strtok('\0'," "));       /* El tercer campo es y */

        /* Hallando la distancia al ascensor 1 */
        Distancia1=pow((x1-xPartida),2)+pow((y1-yPartida),2);
        Distancia1=sqrt(Distancia1);

        /* Busco el ascensor 1ºjerarquia+0+6 */
        Ascensor=(int)(NodoPartida/100)*100+6;
        Nodo=(char *)Buscar_Nodo(Ascensor);

        /* En la variable "nodo" tengo el nodo cuyo nombre coincide con ascensor6*/
        /* Ahora hay que recoger el campo 2º y 3º que son respectivamente la posicion x e
        y */
        x6=atof(strtok('\0'," "));       /* El segundo campo es x */
        y6=atof(strtok('\0'," "));       /* El tercer campo es y */

        /* Hallando la distancia al ascensor 6 */
        Distancia6=pow((x6-xPartida),2)+pow((y6-yPartida),2);
        5Distancia6=sqrt(Distancia6);

        if(((NodoPartida/100)<3)&& (Destino3<3))    /* Entonces hay 4 ascensores: baja,
        1º y 2º planta */
        {
                /* Busco el ascensor 1ºjerarquia+0+2 */
                Ascensor=(int)(NodoPartida/100)*100+2;
                Nodo=(char *)Buscar_Nodo(Ascensor);

                /* En la variable "nodo" tengo el nodo cuyo nombre coincide con ascensor2
                */
                /* Ahora hay que recoger el campo 2º y 3º que son respectivamente la
                posicion x e y */
                x2=atof(strtok('\0'," "));       /* El segundo campo es x */
                y2=atof(strtok('\0'," "));       /* El tercer campo es y */

                /* Hallando la distancia al ascensor 2 */
                Distancia2=pow((x2-xPartida),2)+pow((y2-yPartida),2);
                Distancia2=sqrt(Distancia2);

                /* Busco el ascensor 1ºjerarquia+0+5 */
                Ascensor=(int)(NodoPartida/100)*100+5;
                Nodo=(char *)Buscar_Nodo(Ascensor);

                /* En la variable "nodo" tengo el nodo cuyo nombre coincide con ascensor5
                */
```

```
                /* Ahora hay que recoger el campo 2º y 3º que son respectivamente la
                posicion x e y */
                x5=atof(strtok('\0'," "));       /* El segundo campo es x */
                y5=atof(strtok('\0'," "));       /* El tercer campo es y */

                /* Hallando la distancia al ascensor 5 */
                Distancia5=pow((x5-xPartida),2)+pow((y5-yPartida),2);
                Distancia5=sqrt(Distancia5);
        }


        /* Ahora que tengo las distancias hay que ver cual es el mas cercano */
        if(((NodoPartida/100)<3) && (Destino3<3))   /* Entonces hay 4 ascensores: baja,
        1º y 2º planta */
        {
                /* El ascensor 1 es el mas cercano */

        if((Distancia1<Distancia2)&&(Distancia1<Distancia5)&&
        (Distancia1<Distancia6))
        {
        /* Introduzco en la ruta el nodo jerarquia asociado y el propio ascensor1 */
                        ListaNodos[i++]=(int)(NodoPartida/100)*100;
                        ListaNodos[i++]=(int)(NodoPartida/100)*100+1;

        /* Introduzco en la ruta ahora la salida del ascensor IDEM A ANTERIOR */
                        ListaNodos[i++]=Destino3*100+1;
                        ListaNodos[i++]=Destino3*100;
                 }

                 /* El ascensor 2 es el mas cercano */
                else
if((Distancia2<Distancia1)&&(Distancia2<Distancia5)&&(Distancia2<Distancia6))
                {
                /* Introduzco en la ruta el nodo ascensor2 */
                        ListaNodos[i++]=(int)(NodoPartida/100)*100+2;
                /* Introduzco en la ruta ahora la salida del ascensor IDEM A ANTERIOR */
                        ListaNodos[i++]=Destino3*100+2;
                }

                /* El ascensor 5 es el mas cercano */
                else
if((Distancia5<Distancia1)&&(Distancia5<Distancia2)&&(Distancia5<Distancia6))
                {
                /* Introduzco en la ruta el nodo ascensor5 */
                        ListaNodos[i++]=(int)(NodoPartida/100)*100+5;
                /* Introduzco en la ruta ahora la salida del ascensor IDEM A ANTERIOR */
                        ListaNodos[i++]=Destino3*100+5;
                }

                /* El ascensor 6 es el mas cercano */
                else
if((Distancia6<Distancia2)&&(Distancia6<Distancia5)&&(Distancia6<Distancia1))
                {
                /* Introduzco en la ruta el nodo jerarquia asociado y el propio ascensor6
                */
                        ListaNodos[i++]=(int)(NodoPartida/100)*100+7;
                        ListaNodos[i++]=(int)(NodoPartida/100)*100+6;
                /* Introduzco en la ruta ahora la salida del ascensor IDEM A ANTERIOR */
                        ListaNodos[i++]=Destino3*100+6;
                        ListaNodos[i++]=Destino3*100+7;
                }
        }

        else
        {
                /* El ascensor 1 es el mas cercano */
                if(Distancia1<Distancia6)
                {
                /* Introduzco en la ruta el nodo jerarquia asociado y el propio ascensor1
                */
                        ListaNodos[i++]=(int)(NodoPartida/100)*100;
                        ListaNodos[i++]=(int)(NodoPartida/100)*100+1;
                /* Introduzco en la ruta ahora la salida del ascensor IDEM A ANTERIOR */
                        ListaNodos[i++]=Destino3*100+1;
                        ListaNodos[i++]=Destino3*100;
                }
```

```
                /* El ascensor 6 es el mas cercano */
                else if(Distancia6<Distancia1)
                {
                /* Introduzco en la ruta el nodo jerarquia asociado y el propio ascensor6
                */
                        ListaNodos[i++]=(int)(NodoPartida/100)*100+7;
                        ListaNodos[i++]=(int)(NodoPartida/100)*100+6;
                /* Introduzco en la ruta ahora la salida del ascensor IDEM A ANTERIOR */
                        ListaNodos[i++]=Destino3*100+6;
                        ListaNodos[i++]=Destino3*100+7;
                }
        }

/* Fin de funcion Ascensor_Proximo*/
}

/*-----------------------------------------------------------------*/
// Nombre: Buscar_Mapa
// Funcion:
//     Permite anadir a la lista de nodos de la ruta el camino optimo
//             para salir del mapa actual en busca del mapa del destino
// Parametros entrada:
//             char VbleOrigen              'E' indica ala este
//             char VbleDestino             'O' indica ala oeste
//                                          'N' indica ala norte
//                                          'S' indica ala sur
// Parametros salida:
//             int -> nombre del nodo destino parcial para salir del mapa
//                           actual
/*-----------------------------------------------------------------*/

int Buscar_Mapa(char VbleOrigen, char VbleDestino)
/* En función de la información del mapa donde se encuentra el destino    */
/* esta funcion permite calcular cual es el destino parcial del movil,    */
/* al menos hasta encontrar un nuevo mapa                                 */
{

        float Distancia1,Distancia2;
        float xOrigen,yOrigen,xMapa,yMapa;
        int MapaProximo;
        char *Nodo;

        /* Busco el nodo origen */
        Nodo=(char *)Buscar_Nodo(NodoOrigen);

        /* En la variable "nodo" tengo el nombre del nodo*/
        /* Ahora hay que recoger el campo 2º y 3º que son respectivamente la posicion x e
        y */
        xOrigen=atof(strtok('\0'," "));       /* El segundo campo es x */
        yOrigen=atof(strtok('\0'," "));       /* El tercer campo es y */

        /* Calculo cual es la salida mas proxima del origen, en caso de que ala no sea
        contigua */
        /* Busco el nodo_mapa a la derecha del ala*/
        Nodo=(char *)Buscar_Nodo(Origen3*100);

        /* En la variable "nodo" tengo el nombre del nodo */
        /* Ahora hay que recoger el campo 2º y 3º que son respectivamente la posicion x e
        y */
        xMapa=atof(strtok('\0'," "));       /* El segundo campo es x */
        yMapa=atof(strtok('\0'," "));       /* El tercer campo es y */
        Distancia1=pow((xOrigen-xMapa),2)+pow((yOrigen-yMapa),2);
        Distancia1=sqrt(Distancia1);

        /* Busco el nodo_mapa a la izquierda del ala */
        Nodo=(char *)Buscar_Nodo(Origen3*100+7);

        /* En la variable "nodo" tengo el nombre del nodo */
        /* Ahora hay que recoger el campo 2º y 3º que son respectivamente la posicion x e
        y */
        xMapa=atof(strtok('\0'," "));       /* El segundo campo es x */
        yMapa=atof(strtok('\0'," "));       /* El tercer campo es y */
        Distancia2=pow((xOrigen-xMapa),2)+pow((yOrigen-yMapa),2);
        Distancia2=sqrt(Distancia2);

        if(Distancia2<Distancia1)    MapaProximo=Origen3*100;
        else   MapaProximo=Origen3*100+7;
```

```
        switch(VbleOrigen)
        {
                case 'O':
                switch(VbleDestino)
                {
                        case 'N':       return(Origen3*100+7);
                                        break;
                        case 'S':       return(Origen3*100);
                                        break;
                        case 'E':       return(MapaProximo);
                                        break;
                }
                case 'S':
                switch(VbleDestino)
                {
                        case 'N':       return(Origen3*100+7); /* pej */
                                        break;
                        case 'O':       return(Origen3*100+7);
                                        break;
                        case 'E':       return(MapaProximo);
                                        break;
                }
                case 'E':
                switch(VbleDestino)
                {
                        case 'N':       return(Origen3*100);
                                        break;
                        case 'S':       return(Origen3*100+7);
                                        break;
                        case 'O':        return(MapaProximo);
                                        break;
                }
                case 'N':
                switch(VbleDestino)
                {
                        case 'E':       return(Origen3*100+7);
                                        break;
                        case 'S':       return(MapaProximo);
                                        break;
                        case 'O':       return(Origen3*100);
                                        break;
                }
        }

/* Fin de la Funcion Buscar_Mapa */

}


/*-----------------------------------------------------------------*/
// Nombre: Plan_Ruta
// Funcion:
//      Permite obtener la lista de nodos que conforman la ruta optima
// Parametros entrada: (es char porque puede contener letras)
//      char *Origen -> nombre del nodo origen
//      char *Destino -> nombre del nodo destino
// Parametros salida: (No necesita pues es vble global)
/*-----------------------------------------------------------------*/

void Plan_Ruta(char *Origen, char *Destino)
{
        int NodoDestinoAux;

        /* El primer elemento de la lista es el nodo de salida */
        NodoOrigen=atoi(strtok(Origen,"_"));
        ListaNodos[i++]=NodoOrigen;

        /* Se desmenuza el nombre del nodo origen*/
        Origen3=NodoOrigen/100;
        NodoOrigen = NodoOrigen%100;
        Origen2=NodoOrigen/10;
        NodoOrigen = NodoOrigen%10;
        Origen1=NodoOrigen;
        NodoOrigen=atoi(strtok(Origen,"_"));

        /* ¿Existe cambio de mapa en la trayectoria? */
```

```
        if(Origen[4]!=Destino[4])     /* Esta en otro mapa */
        {       NodoDestino=Buscar_Mapa(Origen[4],Destino[4]);
                /* Simplemente paso nombre mapa quiero ir */
                puts("Cambiando de mapa....");
        }
        else
                NodoDestino=atoi(strtok(Destino,"_"));

        /* Se desmenuza el nombre del nodo destino*/
        Destino3=NodoDestino/100;
        NodoDestinoAux = NodoDestino%100;
        Destino2=NodoDestinoAux/10;
        NodoDestinoAux = NodoDestinoAux%10;
        Destino1=NodoDestinoAux;

        /* salida a nivel 2 (pasillo) de jerarquia */
        if(!(ORIGEN_PASILLO || MISMO_PASILLO))
        {       ListaNodos[i++]=Origen3*100+Origen2*10;
                /* salida de nivel 2 correspondiente */
                /* El baño tratamiento especial esta en pasillo que no existe */
                if(Origen2==6) ListaNodos[i-1]-=30;
        }

        /* cambio de nivel 1 de jerarquia */
        if(Origen3!=Destino3)   Ascensor_Proximo(ListaNodos[i-1]);


        /* entrada en nuevo nivel 2 de jerarquia */
        if(!(DESTINO_PASILLO || MISMO_PASILLO))
        {       ListaNodos[i++]=Destino3*100+Destino2*10;
                /* accedo a nivel 2 correspondiente*/
                /* El baño tratamiento especial esta en pasillo que no existe */
                if(Destino2==6)       ListaNodos[i-1]-=30;
        }

        /* acceso a destino */
        ListaNodos[i++]=NodoDestino;

        /* Ahora hay que introducir los nodos de transicion */
        /* esta funcion actualiza la vble global ruta */
        Introduce_Transicion();

        /* Imprimiendo resultado */
        i=0;
        puts("La ruta encontrada tiene la siguiente lista de nodos:");
        while(Ruta[i]!=NodoDestino)
        {       printf("%d ,", Ruta[i++]);
                getch();
        }
        printf("%d \n", Ruta[i]);
}
```

- *planruta.h*

```
/*****************************************************************/
/*             Planificador de rutas. Fichero cabecera           */
/*             TFC Marta Marron. Navegacion                      */
/*                                                               */
/*             planruta.h            8/4/00                      */
/*****************************************************************/

/*****************************************************************/
/*             Area de librerias                                 */
/*****************************************************************/
#include <string.h>
#include <math.h>
#include <stdio.h>

/*****************************************************************/
/*             Area de constantes                                */
/*****************************************************************/
#define MAX_NODOS_MAPA 200
#define MAX_LONG_NODO 200
#define MAX_LONG_NOMBRE 3
#define MAX_NODOS_RUTA 50
#define MISMO_PASILLO ((Origen2==Destino2)&&(Origen3==Destino3))
#define ORIGEN_PASILLO (Origen2==0)
#define DESTINO_PASILLO (Destino2==0)
#define PASO_SUBRED1_01      1
#define PASO_SUBRED1_22
#define PASO_SUBRED1_32
#define SALA_SUBRED1_01      1
#define SALA_SUBRED1_22
#define SALA_SUBRED1_32
#define SALA_SUBRED2_33
#define CENTRAL_01           3
#define CENTRAL_2            5
#define INTERIOR2_12         5
#define INTERIOR1_12         2



/*****************************************************************/
/*               Area de vbles globales EXTERNAS                 */
/*****************************************************************/
extern FILE *Mapa;
extern int Ruta[MAX_NODOS_MAPA];
```

- *driver.c*

```
/*****************************************************************/
/*            Controlador de Trayectorias                        */
/*             TFC Marta Marron. Navegacion                      */
/*            Del trabajo de Johan Person                        */
/*                                                               */
/*             driver.c              8/8/00                      */
/*                                                               */
/*****************************************************************/

#include <stdio.h>
#include <dos.h>
#include <conio.h>
#include <time.h>
#include <sys/timeb.h>
#include <string.h>

/* Definition of the parallel port address and its registers. */
#define BASE_ADDR 0x378
#define STAT_REG BASE_ADDR + 1
#define CONT_REG BASE_ADDR + 2
#define ADDR_MODE BASE_ADDR + 3
#define DATA_MODE BASE_ADDR + 4

/* Union used for transferring floats using four chars. The f and c part
point to the same memory locations. */
union floatPacket
{
       float f;
       unsigned char c[4];
};

void initPort(void);
int checkTimeout(void);
void clearTimeout(void);

void sendByte(int address, unsigned char data);
char getByte(int address);
void sendInt(int address, unsigned int data);
signed int getInt(int address);
void sendIntArray(int address, int data[], int length);
void getIntArray(int address, int data[], int length);

void writeSemaphore(unsigned char sem, unsigned char value);
unsigned char readSemaphore(unsigned char sem);
void initSemaphores(void);
int catchSemaphore(unsigned char sem);
void freeSemaphore(unsigned char sem);

void memoryTest(void);
void clearMemory(void);

/*****************************************************************/
/*                                Funciones                      */
/*****************************************************************/

/* F1.1 This function initalises the parallel port and the communication card. */
void initPort(void)
{
       /* The following configuration of the Control Register is used:
              C7 reserved, always high.
              C6 reserved, always high.
              C5 low to not disable data outputs.
              C4 low - Enable Interrupt disabled due to problems in Windows.
              C3 high (inverted) nASTRB disactivated.
              C2 low, only high for ECP mode.
              C1 high (inverted) nDSTRB disactivated.
              C0 high (inverted) nWRITE high - default state.
       */
       outportb(CONT_REG, 0xC0);
       clearTimeout();
       initSemaphores(); // Writes "1" to each semaphore.
}
```

```
/* F1.2 This function checks the timeout bit (S0) of the Status Register
and clears it if active. */
int checkTimeout(void)
{
	int timeout;
	timeout = inportb(STAT_REG); // Checks Status Register.

	if (timeout & 1) // Timeout has occured.
	{
		puts("Timeout en escritura DUALPORT!");
		clearTimeout();
		timeout = 1;
	}
	else
		timeout = 0;
	return timeout;
}

/* F1.3 This function clears the timeout bit (S0) of the Status Register using
two methods to cover most EPP chips. */
void clearTimeout(void)
{
	outportb(STAT_REG, 1); // Writes "1" to S0 of Status Register.
	inportb(STAT_REG);     // Reads S0 of Status Register.
}

/* F2.1 This function sends one byte of data to the communication card
using six EPP transfers. The byte may be signed or unsigned. */
void sendByte(int address, unsigned char data)
{
	unsigned char high, low;

	if (address >= 0 && address <= 16383)
	{
		// Separates high and low address bytes.
		high = address / 256;
		low = address % 256;

		outportb(ADDR_MODE, 0);
		outportb(DATA_MODE, low);
		outportb(ADDR_MODE, 1);
		outportb(DATA_MODE, high);
		outportb(ADDR_MODE, 2);
		outportb(DATA_MODE, data);
		checkTimeout();
	}
	else
		puts("Direccion incorrecta en escritura DUALPORT");
}

/* F2.2 This function receives one byte of data from the communication card
using six EPP transfers. The byte may be signed or unsigned. */
char getByte(int address)
{
	unsigned char data, high, low;

	if (address >= 0 && address <= 16383)
	{
		// Separates high and low address bytes.
		high = address / 256;
		low = address % 256;

		outportb(ADDR_MODE, 0);
		outportb(DATA_MODE, low);
		outportb(ADDR_MODE, 1);
		outportb(DATA_MODE, high);
		outportb(ADDR_MODE, 2);
		data = inportb(DATA_MODE);
		checkTimeout();
	}
	else
	{
		puts("Direccion incorrecta en escritura DUALPORT");
		data = -1;            // no me convence, deberia ser un char....
	}
```

```
	return data;
}

/* F2.3 This function sends a 2 byte integer to the communication card.
The int may be signed or unsigned. */
void sendInt(int address, unsigned int data)
{
	unsigned char data_b0, data_b1;

	// Separates high and low data bytes.
	data_b1 = data / 256;
	data_b0 = data % 256;

	sendByte(address + 1, data_b0);
	sendByte(address, data_b1);
}

/* F2.4 This function receives a 2 byte integer from the communication card.
The int may be signed or unsigned. */
signed int getInt(int address)
{
	unsigned char data_b0, data_b1;
	signed int data;

	data_b0 = getByte(address + 1);
	data_b1 = getByte(address);
	data = (signed int)(data_b1 * 256 + data_b0);
	// Merges high and low data bytes.
	return data;
}

/* F2.9 This function sends an array of integers to the communication card.
The integers must be signed. */
void sendIntArray(int address, int data[], int length)
{
	int index;

	for (index = 0; index < length * 2; index = index + 2)
		sendInt(address + index, data[index/2]);
}

/* F2.10 This function receives an array of integers from the communication card.The
integers must be signed. */
void getIntArray(int address, signed int data[], int length)
{
	int index;

	for (index = 0; index < length * 2; index = index + 2)
		data[index/2] = getInt(address + index);
}
/* F3.1 This function attempt to write to a semaphore specified by "sem". */
void writeSemaphore(unsigned char sem, unsigned char value)
{
	if (sem <= 7 && sem >= 0)
	{
		outportb(ADDR_MODE, 0);
		outportb(DATA_MODE, sem);
		outportb(ADDR_MODE, 3);     // Indicate semaphore operation
		outportb(DATA_MODE, value); // Attempt to write semaphore
		checkTimeout();
	}
	else   puts("Direccion de semaforo incorrecta");
}

/* F3.2 This function reads the semaphore specified by "sem". */
unsigned char readSemaphore(unsigned char sem)
{
	if (sem <= 7 && sem >= 0)
	{
		outportb(ADDR_MODE, 0);
		outportb(DATA_MODE, sem);
		outportb(ADDR_MODE, 3);   // Indicate semaphore operation
		sem = inportb(DATA_MODE); // Read semaphore status
		checkTimeout();
	}

	else
```

```
        {
                puts("Direccion de semaforo incorrecta");
                sem=-1;
        }

        return sem;
}

/* F3.3 This function attempts to gain control of a semaphore spcified
by "sem" and returns the result. */
int catchSemaphore(unsigned char sem)
{
        int status;

        writeSemaphore(sem, 0);
        status = readSemaphore(sem);

        if (status == 255)
                status = 1;
                if (status != 1 && status != 0)
        puts ("El semaforo devolvio un valor incorrecto. Posible error en la
                        memoria");
        return status;
}

// F3.4 This function makes the semaphore "sem" available for the Neuron Chip.
void freeSemaphore(unsigned char sem)
{
        writeSemaphore(sem, 1);
}

/* F3.5 This function initialises (resets) the semaphores by writing "1"
to each semaphore. */
void initSemaphores(void)
{
        unsigned char sem;

   for (sem = 0; sem < 8; sem++)
        freeSemaphore(sem);
}

/* F4.1 This function writes and verifies 4 different data at each memory
location in the Dual-Port RAM. In case of errors, these are written to
the file "errors.wri" in the same directory as this program. The test
values are 00000000, 01010101, 10101010 and 11111111. */
void memoryTest(void)
{
        int address, transf = 0;
        unsigned char sent, read;
        int errors = 0;

        puts("Testeando memoria");
        for (address = 0; address <= 16383; address++)
        {
                        transf = (transf + 1)%4;

                        switch(transf)
                        {
                                case 0 : sent = 0; break;
                                case 1 : sent = 85; break;
                                case 2 : sent = 170; break;
                                default : sent = 255;
                        }

                        sendByte(address, sent);
                        read = getByte(address);

                        if (read != sent) errors++;
        }

        if (errors > 0)
                puts(" Ha habido errores en la escritura de la memoria");

}

/* F4.9 This function writes "0" to all memory cells. */
void clearMemory(void)
```

```
{
        int address;

        for (address = 0; address < 16384; address++)
                sendByte(address, 0);
}
```

2. Programas para el NeuronChip.

Los programas para el NeuronChip son los siguientes:

- *epp.nc*: Módulo de comunicación entre el EPP del PC y el NeuronChip.
- *derecho.nc*: Lazo de control PI para la rueda derecha de la silla de ruedas.
- *izquierdo.nc*: Lazo de control PI para la rueda izquierda de la silla de ruedas.

Los módulos *derecho.nc* e *izquierdo.nc* son prácticamente iguales, por tanto sólo se muestra unos de ellos.

- *epp.nc*

```
//////////////////////////////////////////////////////////////////////////
//                                                                      //
//                     Communication between PC and LON                 //
//                      Project work by Johan Persson                   //
//                             15th of Dec 1999                         //
//                                                                      //
//////////////////////////////////////////////////////////////////////////

//////////////////////////////////////////////////////////////////////////
//                                                                      //
//                 Navegacion Autonoma de una Silla de Ruedas           //
//                                                                      //
//             Lazo de Control PI para el control de bajo nivel de      //
//                             una silla de ruedas                      //
//                                                                      //
//                        Marta Marron Romera   Julio de 2000           //
//                                                                      //
//////////////////////////////////////////////////////////////////////////

//////////////////////////////////////////////////////////////////////////
//                   Ficheros Cabecera y declaracion de ctes            //
//////////////////////////////////////////////////////////////////////////
#include <stdlib.h>
#include <status.h>

//////////////////////////////////////////////////////////////////////////
//                    Declaracion de Variables de RED                   //
//////////////////////////////////////////////////////////////////////////

network output unsigned long WDerecha;
network output unsigned long WIzquierda;
network output unsigned long status;
network output unsigned long error;

// Declaration of timer.
mtimer repeating POOLING;

// Definition of semaphores.
typedef struct {
         unsigned short int sem[8];
} *Semaphores;
// Definition of the address location of the semaphores.
const Semaphores semNo = (Semaphores) 0xBFC0;
```

```
// definicion de la zona de datos
unsigned long index;
typedef struct {
     char WRight[2];  // Array of 2 x 1 byte datas 0x0 - 0x1. iNTEGER DE C... AQUI UN LONG
     char WLeft[2];  // Array of 2 x 1 byte datas 0x2 - 0x3. INTEGER DE C--- AQUI LONG
} *Test;
// Definition of start of Dual-Port RAM address.
const Test PCNeuron = (Test) 0x8000;
const Test NeuronPC = (Test) 0x8010;

// otras variables
int reintentando1,reintentando2, tiempo1,tiempo2;
signed long WD_Aux,WI_Aux;

////////////////////////////////////////////////////////////////////////////
//                             Program Events                             //
////////////////////////////////////////////////////////////////////////////

when(reset)
{
         POOLING=10; // Iterval (ms) to enter the "main" process.
         WDerecha=0;
         WIzquierda=0;
         status=0;
         error=0;
         reintentando1=0;
         reintentando2=0;

         // Free semaphores (intialisation).
         for (index = 0; index < 8; index++)
                  semNo->sem[index] = 1;               // libres para quien los quiera
}

when((timer_expires(POOLING)|| reintentando1 || reintentando2))
{
        // VOY A LEER
        // miro a ver si lo tengo
        if(semNo->sem[0]!=0)
            semNo->sem[0]=0;    // si no lo tengo intento cojerlo

        // solo leo si he conseguido el semaforo
        if(semNo->sem[0]==0)
        {   WD_Aux=(signed long)(PCNeuron->WRight[0]*256+PCNeuron->WRight[1]);
            WI_Aux=(signed long)(PCNeuron->WLeft[0]*256+PCNeuron->WLeft[1]);
             reintentando1=0;
        }


        else     // sino activo flag reintentando, para no tener que esperar Ts
        {   reintentando1++;      // asi entra aunqeu no haya pasado el Ts, para leer
            if(reintentando1==1)  // la 1§ vez inicializo el timer
            {   tiempo1=get_tick_count();
            }
            if(get_tick_count()>(tiempo1+2))
            {   reintentando1=0;
                error++;
            }
        }

        // liberando el semaforo..... lo pongo aqui???
        semNo->sem[0] = 1; // Make sem no 4 available for PC side.
```

```
        // VOY A ESCRIBIR
        if (semNo->sem[1] != 0)
            semNo->sem[1] = 0; // Try to gain control of sem no 0  TENGO QUE ESPERAR????

        if (semNo->sem[1] == 0) // If Neuron Chip owns area repr. by sem no 0.
        {       WDerecha=(unsigned long)WD_Aux;
                WIzquierda=(unsigned long)WI_Aux;
                NeuronPC->WRight[0]=(char)(WDerecha/256);
                NeuronPC->WRight[1]=(char)(WDerecha%256);
                NeuronPC->WLeft[0]=(char)(WIzquierda/256);
                NeuronPC->WLeft[1]=(char)(WIzquierda%256);
                reintentando2=0;
         }

        else     // sino activo flag reintentando, para no tener que esperar Ts
        {   reintentando2++;      // asi entra aunqeu no haya pasado el Ts, para leer
            if(reintentando2==1)  // la 1§ vez inicializo el timer
            {   tiempo2=get_tick_count();
            }
            if(get_tick_count()>(tiempo2+2))
            {   reintentando2=0;
                error++;
            }
        }
         semNo->sem[1] = 1; // Make sem no 4 available for PC side.

}
```

- *derecho.nc*

```
////////////////////////////////////////////////////////////////////////
//                                                                    //
//                 Navegacion Autonoma de una Silla de Ruedas         //
//                                                                    //
//             Lazo de Control PI para el control de bajo nivel de    //
//                            una silla de ruedas                     //
//                                                                    //
//                        Marta Marron Romera   Julio de 2000         //
//                                                                    //
////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////
//                  Ficheros Cabecera y declaracion de ctes           //
////////////////////////////////////////////////////////////////////////

#define TS               10  // Ts lectura de encoder y lazo control
#define KI               50  // La constante es  LA INVERSA DE DICHA CTE.
#define KP                5  // La constante proporcional ES TAMBIEN LA INVERSA
#define PWM_MAX         127  // 1/2 del span del PWM
#define BREAKING        500  // retardo antes de meter el freno

#include <stdlib.h>

////////////////////////////////////////////////////////////////////////
//                           Objetos I/O                              //
////////////////////////////////////////////////////////////////////////

IO_4  input  quadrature ENCODER;          // Contador de pulsos en cuadratura
IO_0  output pulsewidth short PWM=0;      // Se¤al PWM, frec 19KHz.
IO_8  output bit ENA_1=FALSE;             // Control del motor
IO_6  input  bit FAULT_1;                 // Motor desactivado hasta reset
IO_7  input  bit FAULT_2;                 // Enables y Faults activos a nivel
                                          // bajo
IO_1  output bit BRAKE=FALSE;             // Frenado de los motores
                                          // FALSE -> Motor frenado
                                          // TRUE -> Motor en movimiento

////////////////////////////////////////////////////////////////////////
//                   Declaracion de Variables de RED                  //
////////////////////////////////////////////////////////////////////////

struct wheels_net
{
    signed long   slWheelP;   // Posicion acumulada en pulsos de Encoder
    unsigned long ulTime;     // Tiempo total
};

struct control_net
{
    signed long slSetWR, slSetWL;
};

// Informa de la realimentacion de posicion, para realizar el deadreckoning
network output  bind_info(unackd)   struct wheels_net  nvo_Real;
struct          wheels_net          struct_Real;

// Informa de la consigna procedente del PC
network input   struct control_net  nvi_SetW;

// Informa de que se ha de parar el sistema por un error.
network input   boolean             nvi_Emergency;
```

```
////////////////////////////////////////////////////////////////////////
/* La unica diferencia entre modulo Derecho e Izquierdo es que de las cuatro
   siguientes lineas hay que comentar:
    En el Izquierdo las dos PRIMERAS
    En el Derecho las dos SEGUNDAS                                      */
////////////////////////////////////////////////////////////////////////
signed  long   *slSetW1=&nvi_SetW.slSetWR;
signed  long   *slSetW2=&nvi_SetW.slSetWL;
//signed  long   *slSetW2=&nvi_SetW.slSetWR;
//signed  long   *slSetW1=&nvi_SetW.slSetWL;

////////////////////////////////////////////////////////////////////////
//                    Declaracion de Variables                        //
////////////////////////////////////////////////////////////////////////

unsigned int  uiActuation;  // Salida para motor tras calculo PID
signed long   slError;      // Acumulacion del error
boolean       bBreaking;    // Indica que el motor esta en proceso de parada
unsigned int  uiMticksAnt;  // Almacena el valor anterior de get_tick_count

////////////////////////////////////////////////////////////////////////
//                       Temporizador Software                        //
////////////////////////////////////////////////////////////////////////

mtimer repeating LOOP1;
mtimer TIMER_BREAKING;

////////////////////////////////////////////////////////////////////////
//                           Prototipos de Funciones                  //
////////////////////////////////////////////////////////////////////////

signed long   EncoderReading(void);
void          PIFilter(signed long);
void          Antiwindup(void);
void          Breaking(void);
unsigned long PeriodMeas(void);
unsigned int  Limit_unsigned_int(signed long);

////////////////////////////////////////////////////////////////////////
//                           Eventos del Programa                     //
////////////////////////////////////////////////////////////////////////

when(reset)
{
    bBreaking=0;                // Desactivado flag de parando
    struct_Real.ulTime=0;       // Inicializo variables de deadreckoning
    struct_Real.slWheelP=0;
    uiActuation=127;            // Inicializacion de consigna
    slError=0;                  // Inicializacion del error

    io_out(BRAKE,FALSE);        // Motor frenado inicialmente
    io_out(PWM, uiActuation);   // Paramos el motor en la primera actuacion.
    io_out(ENA_1,TRUE);         // Para activar el motor

    LOOP1 = TS;                 // Arrancando el Timer para el Ts
}

when(timer_expires(LOOP1))
{
    signed long slEncoder;

    slEncoder=EncoderReading(); // Lectura del encoder
    PIFilter(slEncoder);        // Obtencion de la consigna
```

```
    if(nvi_Emergency==1)        // Si hay emergencia se para el motor
        uiActuation=127;

    io_out(PWM, uiActuation);   // Actuacion de salida por PWM.

    Breaking();                 // Se activa el freno si es necesario

    nvo_Real=struct_Real;       // Se envia el deadreckoning por la red
}

when(timer_expires(TIMER_BREAKING))
{
    io_out(BRAKE, FALSE);
}

when(nv_update_occurs(nvi_Emergency))
{
    if(nvi_Emergency==1)
        slError=0;
}

////////////////////////////////////////////////////////////////////////////
//                                                                        //
//                          Cuerpo de Funciones                           //
//                                                                        //
////////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////////
//                           void Breaking(void)                          //
////////////////////////////////////////////////////////////////////////////

void Breaking(void)
{
// OJO: Para frenar, el objeto 'BRAKE' debe ponerse a cero (FALSE)

    if( (*slSetW1==0) && (*slSetW2==0) && ((uiActuation>=120)&&(uiActuation<=135)) )
    {
       if(!bBreaking)
        {
            TIMER_BREAKING=BREAKING;    // es la 1§ vez que se accede, inicia la cuenta
            bBreaking=1;
        }
    }
    else
    {
        io_out(BRAKE,TRUE);         // Quitando el freno
        bBreaking=0;                // Quito flag de parar;
        TIMER_BREAKING=0;           // Parando el timer;
    }
}

////////////////////////////////////////////////////////////////////////////
//                         void EncoderReading(void)                      //
////////////////////////////////////////////////////////////////////////////

signed long EncoderReading(void)
{
    // Lectura del objeto contador de pulsos en cuadratura
    signed long slEncoder;
    unsigned int uiPeriodo;

    slEncoder = io_in(ENCODER);        // Actualizamos valores ENCODER
    struct_Real.slWheelP += slEncoder; // y de la variable del deadreck
                                              // puede ser negativo el periodo
```

```
    return slEncoder;
}

////////////////////////////////////////////////////////////////////////////
//                            void PIDFilter(void)                        //
////////////////////////////////////////////////////////////////////////////

void PIFilter(signed long slEncoder)
{
    // La consigna ha de llegar en mrad/s
    signed long slControl;
    signed long slConsigna;
    unsigned long=ulPeriod;

    // Calculo y acumulación del error de velocidad.
    // El calculo se realiza en cuentas en un Ts,la consigna se ha de transformar
    ulPeriod=PeriodMeas();
    slConsigna=(*slSetW1);
    slConsigna=muldivs(slConsigna,ulPeriod,1000);   // pasando a pulsos en un Ts
    slConsigna=muldivs(slConsigna,640,314);

    slError += slConsigna - slEncoder;
    slControl = slError/KI + (slConsigna-slEncoder)/KP;

    // Limitacion de la actuacion
    uiActuation = Limit_unsigned_int(slControl);
    Antiwindup();
}

////////////////////////////////////////////////////////////////////////////
//                              PeriodMeas                                //
////////////////////////////////////////////////////////////////////////////

unsigned long PeriodMeas(void)
{
    unsigned int uiMticks;
    unsigned long ulPeriod;

    uiMticks=get_tick_count();
    if(uiMticks<uiMticksAnt)
    {
        uiMticks= (255 - uiMticksAnt + uiMticks + 1);
        struct_Real.ulTime+= uiMticks;
    }

    else
    {
        uiMticks= uiMticks - uiMticksAnt;
        struct_Real.ulTime+= uiMticks;
    }
    uiMticksAnt=uiMticks;
    ulPeriod=muldiv(uiMTicks,TICK_INTERVAL,1000);   // pasando a ms

    return ulPeriod;        // es el periodo
}


////////////////////////////////////////////////////////////////////////////
//                void Antiwindup(void)                                   //
////////////////////////////////////////////////////////////////////////////
void Antiwindup(void)
{
// Si la actuacion esta saturada, limito la integral del error
// al valor correspondiente a esta saturacion:
```

```
    if (uiActuation == 255)
        slError = 127*KI;     // Limite superior
    else if (uiActuation == 0)
        slError = -127*KI;    // Limite inferior
}

////////////////////////////////////////////////////////////////////////////
//         unsigned int Limit_unsigned_int(signed long slActua)           //
////////////////////////////////////////////////////////////////////////////

unsigned int Limit_unsigned_int(signed long slActua)
{
    unsigned int  uiActua;

    slActua+=PWM_MAX;           // Desplazamiento de 127 de la actuacion
    if(slActua<0)
        slActua=0;
    else if(slActua>255)
        slActua=255;

// Comprobacion de paso por PWM 127 (50%)
    if((slActua<127) && (uiActuation>127))
        slActua=127;
    else if((slActua>127) && (uiActuation<127))
        slActua=127;

    uiActua=(unsigned int) slActua;
    return uiActua;
}
```

VI. PRESUPUESTO

En esta sección se estima el importe de la ejecución del proyecto. Para ello se realizará un estudio dividido en diversos apartados, en los cuales se agrupan los gastos según su origen.

1. Coste de los materiales.

En este apartado se engloba el precio del uso de los diversos equipos empleados para desarrollar el presente trabajo, describiendo tanto el precio de la parte hardware como el de la parte software. Por último, se hará un pequeño resumen del conjunto de material de oficina utilizado durante la realización del proyecto.

- Recursos hardware:

EQUIPO	Coste hora	Total horas
1. Silla de ruedas equipada con tarjetas de control de bajo nivel	300 pts.	100 horas
2. Equipo de desarrollo de redes LonWorks (LonBuider)	300 pts.	100 horas
3. Tarjeta interface LonWorks-EPP	150 pts.	100 horas

Coste total de los recursos hardware ..... 75.000 pts.

• Recursos software:

EQUIPO	Coste hora	Total horas
1. Ordenador PENTIUM-200Mhz (con software emulador, Borland C, Matlab, Word y LonBuilder).	200 pts.	800 horas
2. Impresora HP Laserjet 2100	300 pts.	20 horas
3. Impresora HP Laserjet 4500 C	450 pts.	5 horas

Coste total de los recursos software ..... 168.250 pts.

• Material de oficina:

EQUIPO	Coste
1. Material fungible (papel, repuestos impresora, ...)	10.000 pts.
2. Material no fungible	5.000 pts.

Coste total del material de oficina ..... 15.000 pts.

Llegados a este punto, se puede realizar el cálculo final del coste que suponen el conjunto de todos los materiales:

Total de los recursos hardware ..... 75.000 pts.
Total de los recursos software ..... 168.250 pts.
Total de material de oficina ..... 15.000 pts.

Coste total del material ..... 258.250 pts.

El coste total de los materiales asciende a *doscientas cincuenta y ocho mil doscientas cincuenta pesetas*.

2. Coste de la mano de obra.

La realización de este proyecto ha sido llevada a cabo por las siguientes personas:

EQUIPO	Coste hora	Total horas
1. Un Ingeniero en Electrónica redactor y director del proyecto	9.700 pts.	1.500 horas
2. Un mecanógrafo encargado del mecanografiado del proyecto	2.000 pts.	100 horas

Coste total de la mano de obra ..... 14.750.000 pts.

El coste total de la mano de obra asciende a *catorce millones setecientas cincuenta mil pesetas*.

3. Presupuesto de ejecución de material.

Es la suma total de los importes del coste de materiales y de la mano de obra.

Coste total del material ..... 258.250 pts.
Coste total de la mano de obra ..... 14.750.000 pts.

Coste total de ejecución de material ..... 15.008.250 pts.

El presupuesto total de ejecución del material asciende a *quince millones ocho mil doscientas cincuenta pesetas*.

4. Importe de ejecución por contrata.

Se incluyen en este apartado los gastos derivados del uso de las instalaciones donde se ha llevado a cabo el proyecto, cargas fiscales, gastos financieros, tasas administrativas y derivados de las obligaciones de control del proyecto. De igual forma se incluye el beneficio industrial. Para cubrir estos gastos se establece un recargo del 22% sobre el importe del presupuesto de ejecución material.

Coste total de ejecución de material ..... 15.008.250 pts.
22% gastos financieros, beneficios, etc... ..... 330.182 pts.

Coste total de ejecución de material ..... 15.338.432 pts.

El importe de ejecución por contrata asciende a *quince millones trescientas treinta y ocho mil cuatrocientas treinta y dos pesetas*.

5. Honorarios facultativos.

Los honorarios facultativos por la ejecución de este proyecto se determinan de acuerdo a las tarifas de los honorarios de los ingenieros en trabajos particulares vigentes a partir de 1 de septiembre de 1997 dictadas por el Colegio Oficial de Ingenieros de Telecomunicación.

IMPORTE	COEFICIENTE REDUCTOR	PORCENTAJE
Hasta 5 millones	c = 1	7 %
Desde 5 millones	c = 0,9	7 %

Los derechos de visado se calculan aplicando la siguiente fórmula:

$$0{,}07 \times P \times C$$

donde:

P es el presupuesto de ejecución de material, y
C es el coeficiente reductor.

Importe derechos de visado:

0.07 x 5.000.000 x 1 ..... 350.000 pts.
0.07 x 10.008.250 x 1 ..... 700.578 pts.

Total derechos de visado ..... 1.050.578 pts.

El importe total de los derechos de visado es de *un millón cincuenta mil quinientas setenta y ocho pesetas*.

6. Presupuesto total.

El importe del presupuesto total de este proyecto es la suma del presupuesto por contrata y los honorarios facultativos.

Presupuesto por contrata ..... 15.008.250 pts.
Honorarios facultativos ..... 1.050.578 pts.

Presupuesto total ..... 16.058.828 pts.
16% de IVA ..... 2.569.412 pts.

Presupuesto final ..... 18.628.240 pts.

El importe total de este proyecto asciende a la cantidad de DIECIOCHO MILLONES SEISCIENTAS VEINTIOCHO MIL DOSCIENTAS CUARENTA pesetas.

En Alcalá de Henares, a 21 de Septiembre de 2000.

La Ingeniera en Electrónica

Fdo.: Marta Marrón Romera.

VII. BIBLIOGRAFÍA

1. Proyectos fin de carrera.

[1] *"Implementación Hardware de un Controlador de Trayectorias".* Alfredo Gardel Vicente. UPM Escuela Técnica de Ingenieros de Telecomunicación. Junio 1999.

[2] *"Aplicación de Controladores Óptimo y Borroso al Seguimiento de Trayectorias de una Silla de Ruedas".* Elena López Guillén. UA Escuela Politécnica Ingeniería en Electrónica. Marzo 1999.

[3] *"Sistemas de Control de Motores Utilizando Neuron Chip (Parte I)".* Marta Marrón Romera. UA Escuela Politécnica Ingeniería Técnica de Telecomunicación. Octubre 1996.

[4] *"Sistemas de Control de Motores Utilizando Neuron Chip (Parte II)".* Juan Antonio García Cortijo. UA Escuela Politécnica Ingeniería Técnica de Telecomunicación. Junio 1997.

[5] *" Guiado Semiautomático de una Silla de Ruedas".* Eduardo Sebastián Martínez. UA Escuela Politécnica Ingeniería en Electrónica. Octubre 1999.

2. Tesis doctorales y trabajos de tesis.

[6] *"Sistema de Navegación Global Aplicado al Guiado de un Vehículo Autónomo Terrestre en Entornos Exteriores Parcialmente Conocidos".* Miguel Angel Sotelo Vázquez. Diciembre 1999.

[16] *"Seguimiento Facial Mediante Visión Artificial Orientado a la Ayuda a la Movilidad".* Luis Miguel Bergasa Pascual. Diciembre 1999.

[17] *"Contribución al Guiado de un Robot Móvil Utilizando Control Neuronal Adaptativo".* Luciano Boquete Vázquez. Mayo 1998.

3. Artículos de revistas y ponencias en congresos.

[7] *"Automatic Mapping of Dynamic Office Environments".* Clyton Kunz et al. Autonomous Robots Vol.7 Nº2, 131-142. Septiembre 1999. ISBN: 0929-5593

[8] *"The Design of an Autonomous Vehicle for the Disabled".* Richard Madarasz et al. IEEE Journal Robotics and Automation. Vol RA-2, Nº3, 117-126. Septiembre 1986.

[9] *"Visual Recognition of Workspace Landmarks for Topological Navigation".* Panos E. Trahanias el al. Autonomous Robots Vol.7 Nº2, 143-158. Septiembre 1999. ISBN: 0929-5593.

[10] *"A Modular Computig Architecture for Autonomous Robots".* Housheng Hu et al. Microprocessors and Microsystems, Nº 21, 349-361. 1998.

[11] *"Generación de trayectorias y detección de obstáculos mediante splines en el guiado de robots móviles"*. F. Espinosa, M. Mazo, E. López, J. Ureña, F.J. Rodríguez y J.J. García. Centro de Información Tecnológica (CIT), Vol 9, nº 6, págs 125-134. 1998.

[12] *"Sistema de Posicionamiento Absoluto mediante Marcas Artificiales"*. J. C. García el al. TELEC'2000. Junio 2000.

[13] *"Mobile Vehicle Navigation in Unknown Environments: A Multiple Hypothesis Approach"*. D. Maksarov, H. Durrant-Whyte. IEE Control Theory Application. Vol 142, nº 4, 385-400. Julio 1995.

[14] *"Application of Radio Frequency Identification Devices to Support Navigation of Autonomous Robots"*. Olaf Kubitz et al. IEEE 0-7803-3659-3/97. 1997.

4. Documentación de la Web.

[15] *"Vision Based Robotics Using Estimation".* Steven B. Skaar. Tutorial sobre estimación de la Notre Dame University, Indiana. http://www.nd.edu/NDInfo/Research/sskaar/Part2.html. Enero 1994.

5. Otros trabajos.

[18] *"Communication Between PC and LON Using Enhanced Parallel Port".* Johan Persson. Diciembre 1999.

6. Manuales técnicos.

[19] *"LonWorks Technology Device Data".* Motorola 1997.