Índice

I. Resumen

El Trabajo de Fin de Carrera 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 sobre el que se ha trabajado es una silla de ruedas en la que el usuario, proporcionando simplemente el nombre de la sala destino a la que desea ir dentro de un entorno estructurado, se desplaza de forma cómoda al punto deseado.

El objetivo del trabajo consiste en una mejora del sistema de control de posición de la silla haciendo uso de la información obtenida a través de dos subsistemas diferentes: sensado odometríco y visión artificial. Esta información es fusionada mediante un algoritmo denominado Filtro Extendido de Kalman (EKF). Para ello, se han tenido que integrar los trabajos diseñados en varios proyectos realizados anteriormente por otras personas en el marco de la misma linea de investigación y, ademas, se han introducido algunas mejoras generales que se irán detallando a lo largo de este documento.

1. Introducción

1.1. Objetivos perseguidos en el trabajo.

El objetivo final de este proyecto es la integración de varios trabajos para que estos, funcionando de forma conjunta, mejoren el posicionamiento de la plataforma para la que fueron diseñados, esto es, la silla de ruedas autónoma.

Así, partiendo del trabajo [Marrón-00] en el que se conseguía un movimiento autónomo de la silla en un entorno parcialmente estructurado obteniendo la información de posicionamiento relativo gracias a un proceso de *dead reckoning*, se añade un módulo que obtiene información de posicionamiento absoluto gracias a la visión artificial [López-02] y a un sistema de marcas artificiales que se añade al edificio, diseñadas en la tesis [García-01]. Ambos métodos sirven para lo mismo, pero de distintas formas.

Por un lado, la información de odometría está muy contaminada por ruido, pero su obtención es muy rápida. Por el contrario, la información obtenida mediante la visión artificial es muy precisa, pero tiene un tiempo de proceso alto que hace inviable su uso como sistema exclusivo de posicionamiento. Una buena solución a este problema es la desarrollada en [Marrón-02] y será la que se implemente en este trabajo. Ésta consiste en fusionar ambas informaciones mediante un filtro de kalman extendido (EKF). Gracias a este filtro se consigue un equilibrio adecuado entre velocidad de obtención de la información y calidad de la información procesada.

El algoritmo de filtrado del EKF se separa en dos etapas: predicción y corrección. La primera se basará únicamente en la información de odometría y en la historia pasada del sistema, ofreciendo como resultado una estimación de la posición del movil. Mientras tanto, la segunda corregirá los datos obtenidos en la primera fase con la información del subsistema de visión artificial. Como

se dijo anteriormente, se tardará más tiempo en tener información de visión que de odometría. Sin embargo, el EKF se ejecutará con un periodo igual al más bajo de los dos subsistemas, esto es, igual al de odometría. Es por eso que el controlador del movimiento de la silla confíará en la información que obtiene puntualmente del sistema odométrico, ejecutando únicamente la fase de predicción del EKF, hasta que disponga de información del subsistema de visión. En ese momento se ejecutará el EKF de forma completa, primero predicción y luego corrección, consiguiendo así un conocimiento más preciso de la posición de la silla.

El filtro de Kalman tiene unos requisitos de temporización que, si no se cumplen, pueden hacer que no converja y proporcionar resultados incorrectos. Esto llevaría a la silla a moverse hacia algún sitio inesperado, con el peligro que eso conlleva. Una consideración obvia, por tanto, es que la plataforma de cómputo sobre la que funcione el software diseñado debe ajustarse a unos requisitos cercanos en exigencia al tiempo real. Todo el proyecto se ha desarrollado en C sobre Linux por este motivo, ya que este Sistema Operativo ofrece unas prestaciones suficientes para el sistema implementado. Sin embargo, cada uno de los módulos a integrar está diseñado para un entorno distinto de funcionamiento: el navegador original de la silla está programado en lenguaje C sobre Windows ([Marrón-00]), el módulo de visión en C sobre Linux (López-02]) y el Filtro de Kalman está escrito en lenguaje C para Matlab ([Marrón-02]). Por lo tanto, antes de realizar la integración es precisa una migración del software para otras plataformas a C sobre Linux.

Tanto la migración de código como su adaptación y puesta en funcionamiento se desarrollarán a lo largo de esta memoria.

1.2. Organización de la memoria

La memoria está dividida en seis capítulos, a lo largo de los cuales se intenta detallar al máximo posible el diseño y desarrollo del presente proyecto. La precisión 'al máximo posible' no es gratuita: teniendo en cuenta que este proyecto está fuertemente basado, no ya en los resultados, sino en el trabajo completo realizado en dos trabajos fin de carrera, una tesis doctoral y un trabajo de investigación tutelado se hace imposible entrar al detalle en todos y cada uno de los aspectos concretos de las implementaciones realizadas en ellos. Por tanto, a lo largo de toda la memoria se recurrirá mucho al concepto de 'caja negra', remitiendo al lector interesado al desarrollo completo en la memoria del proyecto correspondiente.

En los capítulos 2 y 3 se pretende realizar una introducción al proyecto estableciendo sus bases teóricas y describiendo las aplicaciones implementadas en anteriores trabajos de fin de carrera de las que se parte para llegar a la aplicación final aquí presentada:

- ✓ En el capítulo 2 se hace la explicación teórica del funcionamiento del Filtro Extendido de Kalman (EKF) y se justifica su utilización para la fusión de información de posicionamiento. Después, se pasa a explicar

los modelos del sistema de posicionamiento: primero, desde el punto de vista odométrico y, después, desde el punto de vista de visión.

- ✓ En el capítulo 3 se pasa a describir las aplicaciones previas que sirven de base a la que aquí se presenta: se explica la aplicación de navegación por entornos estructurados basada en odometría, el sistema de obtención de la posición mediante capturas de imagen tomada a través de una cámara de video buscando en el entorno unas marcas artificiales diseñadas al efecto y, por último, se acaba hablando de la simulación del presente trabajo de fin de carrera utilizando la plataforma Matlab que se desarrolló en [Marrón-02].

Los tres capítulos siguientes se ocupan de detallar la implementación completa que se ha abordado en este trabajo. Se tratan temas de migración de plataforma de ejecución de aplicaciones a una común (Linux), integración de las distintas aplicaciones y funcionamiento final de la aplicación diseñada:

- ✓ El capitulo 4 trata de la integración de las aplicaciones de navegación y visión ya mencionadas y de la implementación del filtro EKF de tal forma que cooperen y se alcance el objetivo común perseguido. Se trata el tema del traspaso de las diferentes plataformas software de origen al destino elegido y de la modificación y ampliación del código para permitir la interoperación entre ellas, ya que pasan a ser procesos de una misma y única aplicación.

- ✓ El capítulo 5 se ocupa de los detalles de funcionamiento de la aplicación total implementada. En él se comentan las fases de inicialización, movimiento del robot autónomo y finalización de la aplicación y se explican las técnicas de planificación y comunicación entre procesos empleadas.

- ✓ El capítulo 6 contiene una descripción exhaustiva del modelo de datos y librería de matrices utilizada en esta aplicación que, a buen seguro, será muy útil para quien quiera continuar con este desarrollo posteriormente.

Finalmente, en el capítulo 6 se exponen los resultados obtenidos en las pruebas de funcionamiento del sistema, se comentan las conclusiones pertinentes sobre ellos y se apuntan trabajos que podrían ser abordados en un futuro para seguir mejorando el sistema.

2. Fundamentos Teóricos. Modelo del Sistema.

A lo largo de este apartado se pasan a describir los fundamentos teóricos en los que se basa el presente Trabajo de Fin de Carrera. Se comienza explicando en detalle la teoría del filtro de Kalman extendido que se emplea para la fusión de los datos de posición para, a continuación, pasar a explicar los modelos odométrico y de visión artificial del sistema de posicionamiento.

2.1 Fundamentos teóricos del Filtro de Kalman

El Filtro de Kalman Extendido (EKF) es un estimador óptimo recursivo que permite fusionar medidas de dos sistemas sensoriales, conocido el modelo no lineal de la planta a partir de las medidas de uno de ellos y la magnitud de la covarianza del error que afecta a todas ellas.

El método probabilístico de estimación que se desarrolla a continuación se basa en las siguientes condiciones previas:

- ✓ Se conoce el modelo sensorial que determina la evolución del vector de estados del robot.
- ✓ Este modelo es lineal.
- ✓ Las medidas de los diferentes sensores están afectadas por ruidos blancos (autocorrelación nula y de media cero), gaussianos (función de densidad de probabilidad en forma de campana) e incorrelados entre sí.

Aplicando estas especificaciones al sistema de posicionamiento que se desea implementar se debe comentar que:

- ✓ **Conocimiento del modelo**: El uso de un modelo matemático que defina la evolución del sistema provoca una disminución de fiabilidad debido a los errores de modelado. Sin embargo, la técnica del KF incorpora la ventaja de estimar la evolución del sistema, compararla con la salida real y mejorar su estimación según va ejecutándose (por eso es recursivo).

- ✓ **Condición de linealidad**: Esta condición no suele cumplirse en la práctica totalidad de los sistemas (de hecho no ocurre con el sistema de interés). Sin embargo, es posible utilizar el estimador en un modelo no lineal realizando una linealización del modelo mediante un desarrollo de Taylor alrededor del punto de trabajo, que será el punto de estimación obtenido por el filtro en cada periodo de ejecución. Este tratamiento dará lugar a la extensión del algoritmo de filtrado, denominado EKF, que es el empleado en esta aplicación.

- ✓ La mayor parte de los procesos naturales continuos tienen un **comportamiento normal**[1]. Esta cuestión puede ser justificada físicamente por el hecho de que, usualmente, los ruidos a las medidas obtenidas de procesos naturales continuos se deben a pequeñas fuentes de ruido superpuestas. Matemáticamente, si se suma un conjunto de variables aleatorias independientes suficientemente amplio, el resultado viene descrito mediante una función de densidad de probabilidad (PDF) gaussiana, independientemente de la forma de las PDFs de cada variable del conjunto[2].

- ✓ La eliminación del **valor medio del ruido** que afecta a todas las medidas de posicionamiento se realiza mediante técnicas de calibración sencillas y es fundamental para simplificar el funcionamiento del estimador. Este proceso se describe ampliamente en [Marrón-02] y [García-01].

- ✓ Aunque la condición de **autocorrelación** es más difícil de satisfacer, también suele cumplirse (al menos, en el ancho de banda de trabajo del modelo utilizado). En efecto, así ocurre en este trabajo.

Estas son, básicamente, las aproximaciones asumidas en el desarrollo teórico de la aplicación de estimación de posición. Para una discusión más detallada de su justificación e implicaciones vease [Marrón-02].

2.1.1. Desarrollo del algoritmo recursivo de estimación óptima

La base del KF es la aplicación de la ley de la probabilidad condicional (regla de Bayes) sobre el vector de estado del modelo del sistema. Teniendo en cuenta la consideración de que el sistema es lineal y observable y suponiendo su comportamiento como el de una variable aleatoria única se obtiene un modelo como el siguiente:

[1] La función de densidad de probabilidad gaussiana también es conocida como normal.

[2] Esta afirmación se conoce como teorema del límite central.

$$\begin{aligned} x(t) &= A \cdot x(t) + B \cdot u(t) + w(t) \\ z(t) &= C \cdot x(t) + v(t) = x(t) + v(t) \end{aligned} \quad <2.1>$$

El desarrollo del KF podría partir del hecho de que en un instante t_1 se predice, a través de su modelo, el comportamiento del sistema (variable aleatoria), con un resultado de estado (o de salida, tal y como se presenta en la ecuación <2.1>) caracterizado por una PDF de media x_1 y varianza σ_1^2. En el instante t_2 se mide dicho comportamiento mediante un sensor que proporciona solamente información de salida, con un resultado de media z_2 y varianza σ_v^2 (incorrelado con el anterior). La probabilidad condicional de que el comportamiento sea el estimado (x_1) midiendo z_2 se obtiene mediante la mencionada regla de Bayes, obteniéndose como resultado una PDF gaussiana de media x_2 y varianza σ_2^2 (ver figura 2.1):

$$\mu = x_2 = \left[\frac{\sigma_v^2}{(\sigma_1^2 + \sigma_v^2)}\right] x_1 + \left[\frac{\sigma_1^2}{(\sigma_1^2 + \sigma_v^2)}\right] z_2$$

$$\frac{1}{\sigma_2^2} = \frac{1}{\sigma_v^2} + \frac{1}{\sigma_1^2} \quad <2.2>$$

La mejor estimación de dicho comportamiento es, por tanto, μ (los ruidos son normales y de media nula) que se podrá poner como la estimación óptima a posteriori, y que, agrupando términos, supondrá:

$$x_2 = x_1 + \left[\frac{\sigma_1^2}{\left(\sigma_1^2 + \sigma_v^2\right)}\right](z_2 - x_1) \quad \Rightarrow$$

$$x_2 = x_1 + K_2(z_2 - x_1), \quad K_2 = \left[\frac{\sigma_1^2}{\left(\sigma_1^2 + \sigma_v^2\right)}\right] \quad <2.3>$$

De esta ecuación se obtiene, por tanto, la matriz de Kalman (K_2) a partir de la varianza del error de estimación σ_1^2 y de la varianza del ruido de medidas σ_v^2. La mejor estimación se obtendrá corrigiendo la obtenida del modelo del sistema (x_1) mediante un término (z_2 - x_1) que se denomina residuo y que está ponderado por la matriz de estimación óptima K_2.

Este sencillo ejemplo muestra cómo el uso de la probabilidad condicional permite que la incertidumbre de estimación se vaya reduciendo ($\sigma_2^2 < \min\left[\sigma_v^2, \sigma_1^2\right]$), según la ecuación <2.2>, al incorporar recursivamente más información al proceso de estimación. Además, se puede obtener otra conclusión de este análisis y es que, por muy mala que sea la medida (σ_v^2), aporta una mejora en cualquier caso a la estimación final de la variable de interés (ver figura 2.1).

También se puede observar en este planteamiento básico el modelo predictor-corrector del algoritmo: la salida estimada será inicialmente la

predicha por el modelo (x_1), que será corregida (x_2) por el factor de residuo (z_2-x_1) y ponderada por la ganancia de Kalman (K2) en el instante siguiente (t2).

Se puede hacer un último análisis con respecto a este ejemplo base. Rescribiendo la ecuación de la varianza presentada en la ecuación <2.2> mediante el uso de la matriz de estimación (K_2) se puede poner:

$$\sigma_2^2 = \sigma_1^2 - K_2\sigma_1^2 \quad <2.4>$$

De esta forma se observa cómo la matriz de varianza de error de predicción se va actualizando también gracias a la matriz de Kalman.

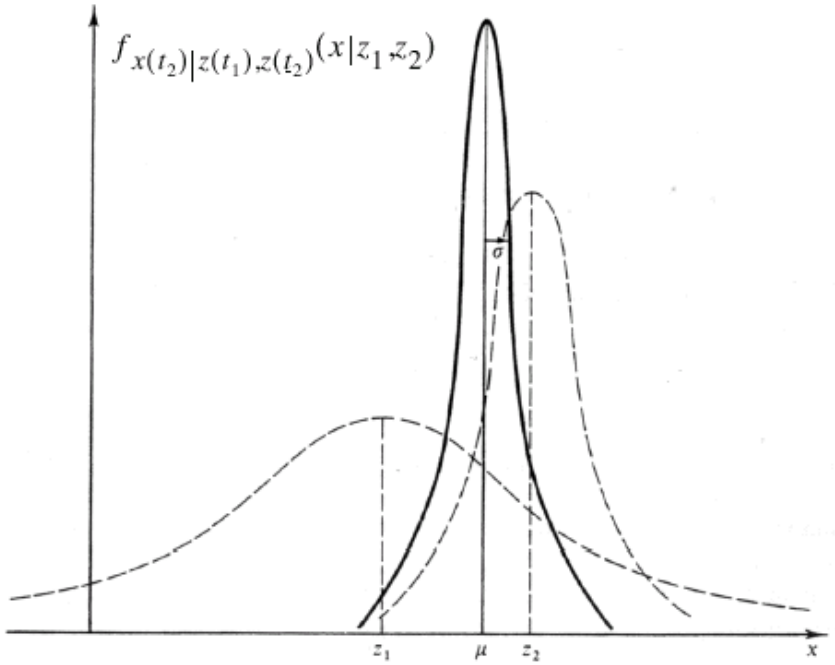


Figura 2.1. Representación gráfica del funcionamiento del KF como estimador basado en el modelo del ruido que afecta a las medidas

El algoritmo puede ser expresado matemáticamente de forma tan sencilla gracias a las condiciones impuestas a las fuentes de ruido que se incluyen en el vector de estados (ruido del sistema) y de medida (ruido de medida). En caso de que las fuentes de ruido no fuesen gaussianas o blancas, de media nula e incorreladas entre sí, el algoritmo se complicaría. Para resolver este problema se han planteado diferentes técnicas que el lector interesado puede encontrar en [Marrón-03].

2.1.2. El algoritmo del KF para sistemas discretos

Para este caso se definirán unos vectores de estado y de medidas multivariables y, con ello, matrices de covarianza en lugar de varianzas (ruido blanco y gaussiano). Para hacer el caso más general, se partirá del modelo lineal completo del sistema discreto:

$$\begin{matrix} \overrightarrow{x_k} = A \cdot \overrightarrow{x_{k-1}} + B \cdot \overrightarrow{u_k} + \overrightarrow{w_k} \\ \overrightarrow{z_k} = C \cdot \overrightarrow{x_k} + \overrightarrow{v_k} \end{matrix}, \quad \begin{matrix} p(\overrightarrow{w}) = N(0,Q) \\ p(\overrightarrow{v}) = N(0,R) \end{matrix} \quad <2.5>$$

Aunque en el caso general se establece que las matrices A, B, C, Q y R son constantes (sistemas invariantes y ruidos estacionarios) la extrapolación del algoritmo a matrices variables no sería muy complejo, por lo que se plantea el caso más simple.

Para comprender la nomenclatura referida a las dos etapas de evolución del algoritmo (predicción y corrección) se definirá $m_{k+1/k}$ como la estimación de la variable m en el instante k+1 a partir de la información existente en el instante k (predicción), y $m_{k+1/k+1}$ como la estimación de la variable m en el instante k+1 a partir de la información existente en el instante k+1 (corrección).

El primer paso en la evolución del algoritmo consiste en definir la matriz de covarianza del error de estimación P, que es la que ha de minimizarse para conseguir un estimador óptimo, como:

$$\begin{matrix} P_{k+1/k} = E\left[\overrightarrow{e_{k+1/k}} \cdot \overrightarrow{e_{k+1/k}}\right] \\ P_{k+1/k+1} = E\left[\overrightarrow{e_{k+1/k+1}} \cdot \overrightarrow{e_{k+1/k+1}}\right] \end{matrix}, \quad \begin{matrix} \overrightarrow{e_{k+1/k}} = \overrightarrow{x_{k+1}} + \hat{x}_{k+1/k} \\ \overrightarrow{e_{k+1/k+1}} = \overrightarrow{x_{k+1}} + \hat{x}_{k+1/k+1} \end{matrix} \quad <2.6>$$

La diferencia entre $\overrightarrow{x_k}$ y $\hat{x}_k$ es que, mientras que la primera se refiere al valor del vector de estado real del sistema, la segunda lo hace a la predicción que se realiza del vector de estado con el KF. En la aplicación del KF a tareas de fusión el significado cambia ligeramente, tal y como se verá más adelante.

Con todo ello, y siguiendo el razonamiento Bayesiano antes presentado (ver ecuación <2.3>), la estimación a posteriori (corrección) del vector de estado se obtendrá del siguiente modo:

$$\hat{x}_{k+1/k+1} = \hat{x}_{k+1/k} + K_{k+1}(z_{k+1} - C \cdot x_{k+1/k}) \quad <2.7>$$

La ganancia de Kalman se calcula minimizando la matriz de covarianza del error de estimación $P_{k+1/k+1}$. Así, se obtiene el siguiente valor:

$$K_{k+1} = \frac{P_{k+1/k} \cdot C^T}{C \cdot P_{k+1/k} \cdot C^T + R} \quad <2.8>$$

Se observa claramente que la ecuación <2.8> tiene un aspecto semejante al presentado en la anterior <2.3>, cambiando la varianza del error de estimación por la matriz de covarianza $P_{k+1/k}$ y añadiendo la matriz C, que antes suponíamos la unidad.

Finalmente, la matriz de covarianza del error de estimación de estados se obtiene a partir de las expresiones anteriores:

$$P_{k+1/k+1} = P_{k+1/k} - K_{k+1} \cdot C \cdot P_{k+1/k} \quad <2.9>$$

Con todo esto se completa el algoritmo buscado. En la siguiente figura se presenta un resumen de las dos etapas básicas comentadas en las que se desarrolla el algoritmo: predicción y corrección.

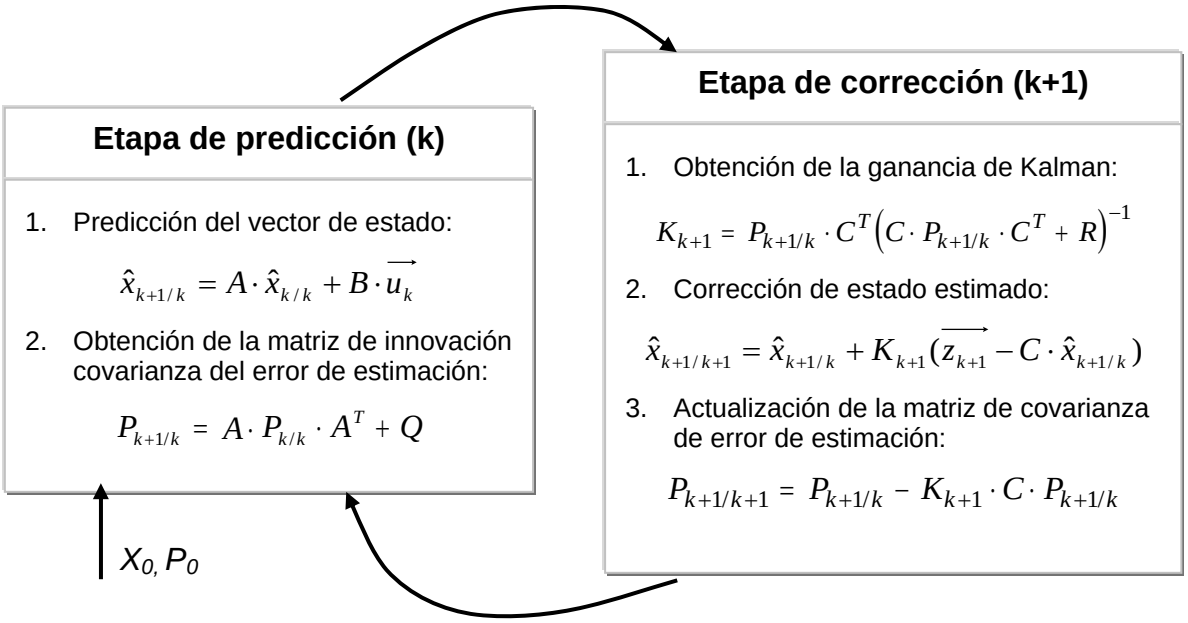


Figura 2.2. Diagrama explicativo del desarrollo del Filtro de Kalman en dos etapas (KF)

En la etapa de predicción (en el instante k) se obtiene el valor del vector de estados estimado ($x_{k+1/k}$) empleando para ello el modelo matemático del sistema. Así se obtendrá la predicción del vector de estado y, además, se obtiene el valor de innovación de la matriz de covarianza del error del predicción o estimación ($P_{k+1/k}$), que implica a la matriz de covarianza de error del sistema (Q).

La etapa de corrección, que se desarrolla en el instante siguiente (k+1), comienza con la obtención de la ganancia de Kalman a partir de las matrices de covarianza de error de estimación innovada ($P_{k+1/k}$) y de covarianza del error de medidas (R). Con ella se corrige la estimación del estado ($\hat{x}_{k+1/k+1}$) del sistema y, finalmente, se actualiza la matriz de covarianza del error de estimación ($P_{k+1/k+1}$). Este último paso consiste fundamentalmente en fusionar la información obtenida con el modelo del sistema con la información externa obtenida del sensado de la salida. Ésto lo que da pie a emplear el algoritmo en procesos de fusión.

Con estos datos se volvería a realizar el proceso de predicción, ahora en el instante k+1 para obtener la estimación del instante k+2, y así consecutivamente.

La recursividad del algoritmo evita tener que realizar cálculos de estimación con toda la batería de datos de instantes anteriores, tal y como

ocurre en otras técnicas de filtrado. De este modo se agiliza en gran medida el algoritmo de estimación o filtrado, haciéndolo válido para aplicaciones en Tiempo Real como las de navegación o posicionamiento de robots móviles.

Tras este desarrollo quedaría totalmente detallado el funcionamiento del estimador óptimo. Merece la pena hacer hincapié en la inicialización del algoritmo que, tal y como se observa en el diagrama de la figura 2.2, pasa por la inicialización de el vector de estado (x_0) y de la matriz de covarianza del error de predicción (P_0). Si bien la primera suele ser fácil de fijar, ya que en las aplicaciones de navegación este vector está relacionado con la posición inicial del mismo (que, en principio, se supone conocida), la segunda no lo es tanto. Hay varias tácticas para asignarle un valor inicial.

- ✓ Si se conoce x_0, P_0 será nulo, ya que representa la incertidumbre de la estimación inicial.

- ✓ Si no se conoce x_0, P_0 deberá fijarse en función de su significado, pero en cualquier caso una técnica muy recurrida es la de ejecutar el estimador off-line (mediante una simulación) y ajustarlo así por prueba y error.

- ✓ Existen métodos matemáticos que permiten obtener el valor final del P_0, pero solamente válidos para casos en los que R y Q son constantes. En estos casos el KF converge fácilmente independientemente del valor inicial de P.

En cualquier caso, si el sistema es lineal e invariante el estimador recursivo suele converger siempre (de forma más o menos rápida) independientemente del valor de P_0. La elección de este parámetro será más crítica a la hora de implementar el estimador para sistemas no lineales.

En lo que a las matrices Q y R se refiere, al ser la covarianza del error de sistema y de medidas, su valor se puede obtener fácilmente (al menos, de forma aparente) ($Q = E\left[\overrightarrow{w_k} \cdot \overrightarrow{w_k^T}\right], R = E\left[\overrightarrow{v_k} \cdot \overrightarrow{v_k^T}\right]$. Aunque v_k y w_k aparecen dependientes del tiempo Q y R suelen considerarse constantes. Sin embargo, si bien la matriz R puede obtenerse de forma empírica realizando medidas off-line de la salida y obteniendo de ellos dicho parámetro estadístico, no ocurre de igual modo con la matriz Q ya que no suele ser habitual tener acceso a las variables de estado del sistema. Afortunadamente, esto sí ocurre en las aplicaciones de fusión, como se explicará más adelante. En caso de no poder obtener Q de forma matemática se fijará por tanteo, generalmente, también mediante sucesivas pruebas del estimador realizadas off-line. En ese caso es importante tener en cuenta ciertos aspectos sobre el valor de estas matrices.

- ✓ La relación de magnitud entre Q y R es indicativa de la fiabilidad de los dos elementos de medida o modelado. Así, si Q<R significa que el modelo proporciona más fiabilidad sobre el vector de estado que la matriz de salida. En ese caso, la matriz P y la ganancia de Kalman tomarán valores pequeños. Si tenemos que R<Q significará que el

modelo es menos fiable que los sensores de salida y, por tanto, P y K serán grandes en la evolución del algoritmo.

- ✓ Ambas matrices son cuadradas y diagonales ya que el ruido no ha de estar autocorrelado, tiene que ser blanco. En caso de que los experimentos de obtención de las matrices devuelvan algún valor no nulo fuera de la diagonal principal el ruido será coloreado y será necesario emplear alguna técnica para corregir este aspecto. Por otro lado, si alguna de las fuentes de ruido presenta fuentes sistemáticas la media de estas variables aleatorias no será nula, por lo que será necesario eliminarlos de algún modo.

- ✓ Finalmente, las dos fuentes de ruido (v_k y w_k) han de estar incorreladas entre sí. Esto suele no cumplirse en aplicaciones de fusión en las que los elementos de sensado son conjuntos o están relacionados de algún modo. Si esto fuera así sería necesario desacoplarlos previamente para poder implementar el KF. En la aplicación descrita en este documento las fuentes de ruido están incorreladas.

2.1.3. El algoritmo del EKF para sistemas discretos y no lineales

El EKF surge de la necesidad de aplicar el estimador óptimo de Kalman a sistemas regidos por un modelo no lineal. La idea básica de la implementación de esta nueva versión del estimador consiste en linealizar el algoritmo alrededor del punto de trabajo mediante una serie de Taylor.

El análisis de funcionamiento del EKF sigue los mismos pasos que el presentado en el punto anterior, sólo que el modelo de partida del sistema sensorial a observar cambia ya que, en este caso no es lineal. Así, será necesario realizar ciertas transformaciones previas para seguir el mismo planteamiento.

La representación general del modelo del sistema queda ahora:

$$\begin{aligned} \vec{x}_k &= f(\vec{x}_{k-1}, \vec{u}_k, \vec{w}_k) \\ \vec{z}_k &= f(\vec{x}_k, \vec{v}_k) \end{aligned} \quad <2.10>$$

Para linealizar el modelo primeramente hay tener en cuenta que en la ecuación <2.10> no se conoce el valor instantáneo del ruido que le afecta, por lo que no se puede incluir en el modelo de estimación y ha de rescribirse del siguiente modo:

$$\begin{aligned} \tilde{x}_k &= f(\hat{x}_{k-1}, \vec{u}_k, 0) \\ \tilde{z}_k &= f(\tilde{x}_k, \vec{v}_k) \end{aligned} \quad <2.11>^{3}$$

[3] De aquí en adelante se suprime la notación vectorial, debiendo entenderse que en cualquier referencia a las variables éstas son vectores.

Linealizando el sistema de ecuaciones anterior queda:

$$\begin{aligned} x_k &= \tilde{x}_k + A_k \cdot \left(x_{k-1} - \hat{x}_{k-1}\right) + W_k \cdot w_{k-1} \\ z_k &= \tilde{z}_k + C_k \cdot \left(x_k - \tilde{x}_k\right) + V_k \cdot v_k \end{aligned} \quad <2.12>$$

Donde:

- ✓ x_k y z_k son los valores y medidas reales del vector de estado y de salida, respectivamente.
- ✓ $\tilde{x}_k$ y $\tilde{z}_k$ son los valores estimados sin error (procedentes del modelo) del vector de estado y de salida, respectivamente.
- ✓ $\hat{x}_k$ es el valor del vector de estado estimado y corregido a posteriori ($\hat{x}_{k+1/k}$).
- ✓ A_k es el jacobiano de derivadas parciales de $f()$ con respecto a x_k en el instante k.

$$A_k \Rightarrow A_{[i,j]} = \frac{\partial f_{[i]}}{\partial x_{[j]}}\left(\hat{x}_{k-1}, u_k, 0\right) <2.13>$$

- ✓ C_k es el jacobiano de derivadas parciales de $h()$ con respecto a x_k en el instante k.

$$C_k \Rightarrow C_{[i,j]} = \frac{\partial h_{[i]}}{\partial x_{[j]}}\left(\hat{x}_k, u_k, 0\right) <2.14>$$

- ✓ W_k es el jacobiano de derivadas parciales de $f()$ con respecto a w_k en el instante k.

$$W_k \Rightarrow W_{[i,j]} = \frac{\partial f_{[i]}}{\partial w_{[j]}}\left(\hat{x}_{k-1}, u_k, 0\right) <2.15>$$

- ✓ V_k es el jacobiano de derivadas parciales de $h()$ con respecto a v_k en el instante k.

$$V_k \Rightarrow V_{[i,j]} = \frac{\partial h_{[i]}}{\partial v_{[j]}}\left(\hat{x}_k, 0\right) <2.16>$$

Las matrices A y C que eran constante en el KF básico no lo son en esta nueva formulación (será necesario recalcular su valor en cada paso de ejecución del estimador óptimo).

A partir de este planteamiento se redefine el error de predicción del vector de estados y de salida de esta forma:

$$\begin{aligned} e_{x_k} &= x_k - \hat{x}_k \\ e_{z_k} &= z_k - \hat{z}_k \end{aligned} \quad <2.17>$$

Además, como no se tiene acceso directo al vector de estados real pero sí al de salida real, empleando las expresiones de la ecuación <2.12> se puede rescribir lo anterior como:

$$\begin{aligned} e_{x_k} &= A_k\left(x_{k-1} - \hat{x}_{k-1}\right) + \varepsilon_k \\ e_{z_k} &= C_k \cdot e_{x_k} + \eta_k \end{aligned} \quad <2.18>$$

Aquí ε_k y η_k son dos nuevas variables aleatorias con media nula y varianza de valor $\sigma_{\varepsilon_k} = W_k Q W_k^T$ y $\sigma_{\eta_k} = V_k R V_k^T$, dado que v_k y w_k lo son.

La ecuación <2.18> anterior tiene un formato semejante al de descripción del modelo de un sistema lineal (ver ecuación <2.5>), por lo que se va a tomar como punto de partida para desarrollar el EKF. En este caso el vector de "estado" es el error de estimación del sistema e_{x_k} que, además, se desea hacer nulo. Con todo ello la ecuación de regulación de este segundo estimador óptimo de Kalman será:

$$\hat{e}_{x_k} = K_k \cdot e_{z_k} \quad <2.19>$$

La ecuación de obtención de la estimación a posteriori del vector de estado original se define ahora como:

$$\hat{x}_k = \tilde{x}_k + \hat{e}_{x_k} \quad <2.20>$$

Sustituyendo la ecuación <2.19> en ésta última, se obtiene ésta de una forma más familiar (semejante a la ecuación <2.7>):

$$\hat{x}_k = \tilde{x}_k + K_k \cdot e_{z_k} = \tilde{x}_k + K_k\left(z_k - \tilde{z}_k\right) \quad <2.21>$$

Para completar el EKF se definen la ganancia del estimador K_k de modo que minimice el error de predicción de e_{k_x} (el vector de estado en este nuevo modelo transformado), tal y como se presentó en la anterior ecuación <2..8>; y la matriz de covarianza del error de estimación P a partir de las anteriores, tal y como se presentaba en la anterior ecuación <2.9>. En este caso, en la expresión de las dos matrices comentadas se sustituye R y Q por sus equivalentes, obtenidas a partir de la expresión <2.18>.

Al igual que en el apartado anterior, en la figura siguiente se presenta un resumen del funcionamiento del algoritmo.

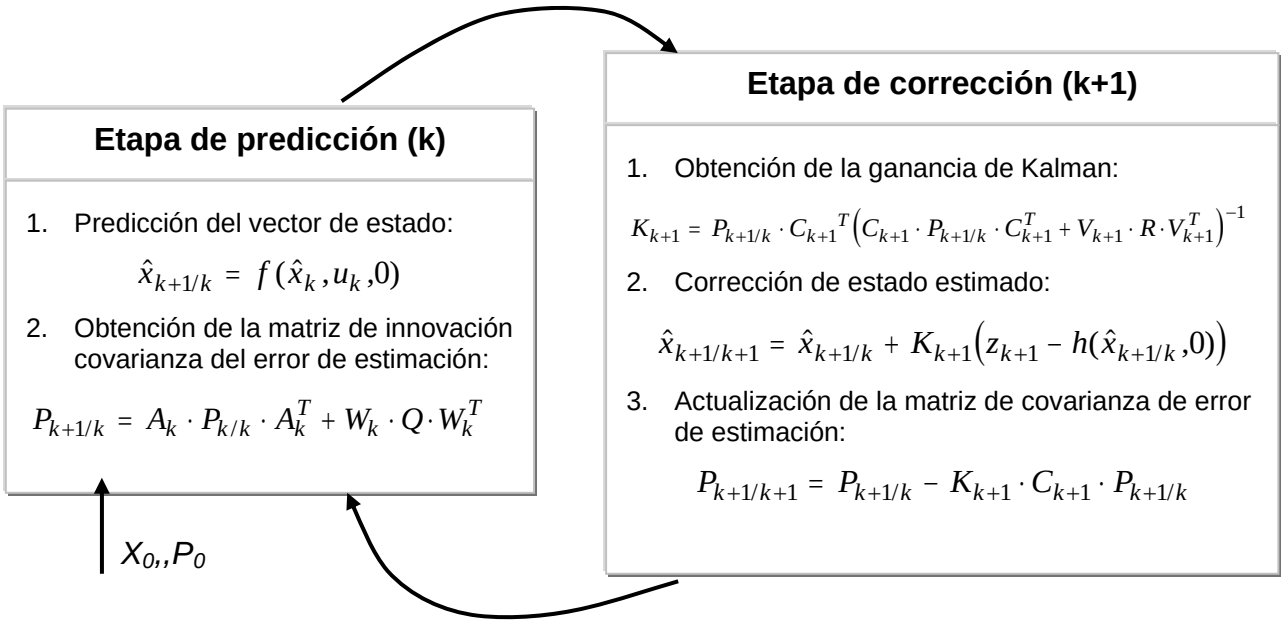


Figura 2.3. Diagrama explicativo del desarrollo del Filtro de Kalman Extendido (EKF)

Para mantener la presencia del concepto de estimación a priori y a posteriori, en las ecuaciones de la figura anterior se ha sustituido $\tilde{x}_k$ por $\hat{x}_{k+1/k}$. Además, se puede observar fácilmente en estas expresiones como A_k, C_k, W_k y V_k ahora sí que se han hecho dependientes del tiempo. Debido a que su origen es no lineal será necesario calcular su valor en cada instante mediante una serie de Taylor centrada en el punto de trabajo.

Es interesante observar que, a la hora de linealizar el modelo del sistema, se ha aplicado también la transformación a las fuentes de ruido que modificaban el vector de estados y de salida (w_k y v_k) obteniendo otras fuentes de ruido (ε_k y η_k). El problema es que esta transformación ha eliminado el carácter gaussiano de las fuentes primitivas, con lo que el planteamiento Bayesiano que da lugar a todo el desarrollo del estimador óptimo ya no es todo lo exacto que se desea. Este hecho produce que el EKF no converja con la misma facilidad que lo hace el KF sino que lo hará dependiendo del valor inicial de la matriz de covarianza del error de estimación P o del modelo del sistema en general. Existen algoritmos derivados del EKF que tienen en cuenta este hecho, como se explica en [Marrón-02].

2.2. Modelos del Sistema

En este apartado se presenta el modelo del sistema sobre el que se desea realizar la tarea de fusión. Para la aplicación bajo estudio han de ser modelados tanto la representación de los estados del sistema de posicionamiento (que se asociarán al sistema odométrico), como la salida en posición que se obtiene mediante el sistema de visión, así como los ruidos asociados a ambas medidas

El sistema inercial será el elegido para determinar la evolución de los estados, por lo que a partir de estas medidas se construirá el modelo cinemático del robot mediante un proceso de dead-reckoning. El modelo obtenido es no lineal, lo cual obliga a hacer diversas consideraciones en su uso para la implementación del algoritmo de fusión y nos lleva al uso de la versión extendida del KF (EKF). Además, es necesario tener en cuenta que la implementación digital del estimador precisa de la obtención de modelos discretizados con un determinado periodo de muestreo.

En lo que al modelo de visión se refiere se mostrará cómo, a partir de la información obtenida por el algoritmo SPL [García-01], se obtiene la salida del modelo de posición del robot mediante una sencilla transformación. Esta información será incorporada durante el proceso de fusión a la estimada a partir de las medidas de odometría, tal y como se mostró en el desarrollo del EKF.

Con esta presentación y lo estudiado en el apartado anterior ya se puede vislumbrar cómo se realizará la fusión de ambos sistemas de medidas mediante un EKF (ver figura 2.4). El elemento inercial, mediante el modelo de dead-reckoning no lineal, permitirá obtener los estados estimados del sistema robótico y, a partir de estos, las salidas de posición del mismo. En este caso, éstas van a coincidir ya que las variables del vector de estado serán las mismas que conforman el vector de salida, como se ve en la figura 2.4. Por otro lado, el sistema de visión proporcionará, mediante la transformación mencionada, la salida que corregirá el estado estimado a través de la ganancia de Kalman (calculada para minimizar la covarianza de error de estimación).

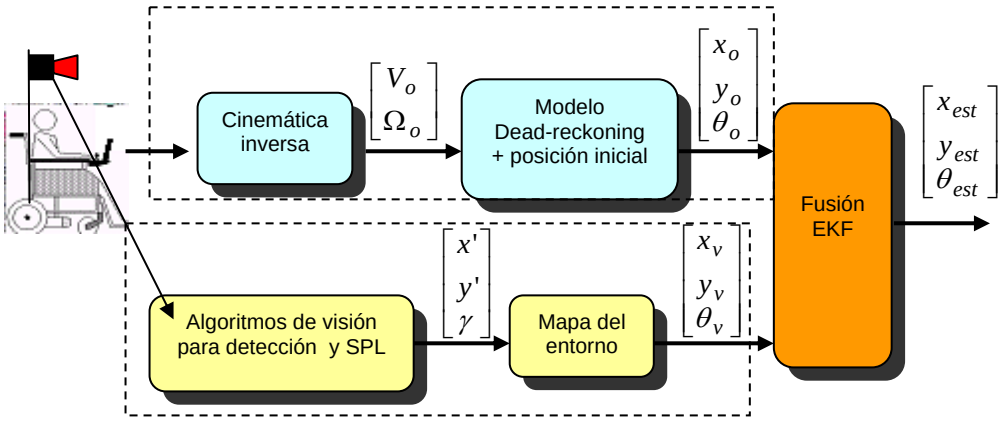


Figura 2.4[4]. Diagrama funcional del algoritmo de fusión para posicionamiento de un robot móvil mediante un EKF

2.2.1 Modelo Odométrico del Sistema

[4] Se utiliza la notación subíndice 'o' para indicar que la información se obtiene por odometría, mientras que el subíndice 'v' para indicar que es información obtenida mediante el sistema de visión.

El modelo inercial del robot móvil objeto de la aplicación de fusión ha sido desarrollado en múltiples trabajos previos [Marrón-00]. La posición del mismo se obtiene mediante un proceso de integración de las medidas de odometría (ver figura 2.4) tal y como se reproduce a continuación:

$$\begin{aligned} \dot{x}_o &= V \cdot \cos\theta_o \\ \dot{y}_o &= V \cdot \sin\theta_o \\ \dot{\theta}_o &= \Omega \end{aligned} \quad \text{, donde} \quad \begin{aligned} V &= \frac{R}{2}\left(\omega_D + \omega_I\right) \\ \Omega &= \frac{R}{D}\left(\omega_D - \omega_I\right) \end{aligned} \qquad <2.22>$$

En la ecuación 2.22 anterior todas las variables son continuas. El sistema de la izquierda constituye específicamente el modelo inercial de la posición del robot, y el de la derecha las ecuaciones de cinemática diferencial inversa de la silla de ruedas.

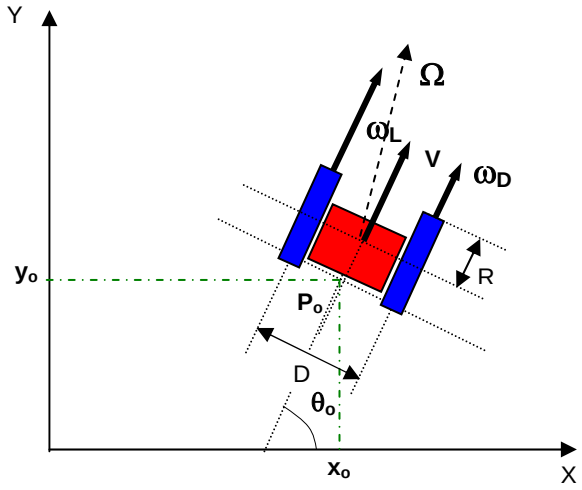


Figura 2.5. Diagrama explicativo de la cinemática diferencial del robot móvil de interés

El modelo obtenido se ha supuesto libre de ruidos, ya que estos serán caracterizados más adelante mediante la correspondiente matriz de covarianza. Además, como el modelado se realiza con el objetivo de poder aplicarlo al estimador óptimo, deberá ser discretizado.

A la vista del modelo matemático, se proponen como entradas del modelo discreto el vector de medidas de velocidad V y Ω, que se obtienen del sistema de odometría mediante la relación cinemática simple que se mostró en la ecuación <2.22>, y como salidas el vector de posición del robot en el plano de movimiento (XY) y la orientación del mismo respecto al eje X ($\vec{z}_{o,k} = \begin{bmatrix} x_{o,k} & y_{o,k} & \theta_{o,k} \end{bmatrix}^T$). Los estados del sistema serán aquellos que determinan la evolución de la posición del mismo y coinciden, en este caso, plenamente con las salidas elegidas ($\vec{x}_{o,k} = \begin{bmatrix} x_{o,k} & y_{o,k} & \theta_{o,k} \end{bmatrix}^T$). El diagrama de la figura 2.6 muestra la organización esquemática del modelo resultante.

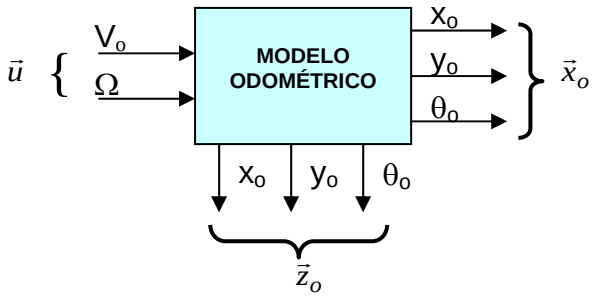


Figura 2.6.. Elección de las variables de interés del modelo odométrico del robot móvil

Con estas consideraciones. el modelo inercial lineal y discreto de la planta vendría descrito por las siguientes ecuaciones de estado y de salida:

$$\begin{bmatrix} x_{o,k} \\ y_{o,k} \\ \theta_{o,k} \end{bmatrix} = A \cdot \begin{bmatrix} x_{ok-1} \\ y_{o,k-1} \\ \theta_{ok-1} \end{bmatrix} + B \cdot \begin{bmatrix} V_k \\ \Omega_k \end{bmatrix}$$

$$\begin{bmatrix} x_{o,k} \\ y_{o,k} \\ \theta_{o,k} \end{bmatrix} = C \cdot \begin{bmatrix} x_{o,k-1} \\ y_{o,k-1} \\ \theta_{o,k-1} \end{bmatrix} + D \cdot \begin{bmatrix} V_k \\ \Omega_k \end{bmatrix} \quad \text{<2.23>}$$

Aquí se aprecia claramente que $D = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}$ y $C = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$

Sin embargo, comparando las ecuaciones <2.22> y <2.23> es también fácil de apreciar que no existe una relación lineal entre las componentes del vector de estado, por lo que no es posible obtener las matrices A y B del modelo. Es necesario, por tanto, hacer un estudio más exhaustivo del modelo odométrico.

2.2.1.1. Modelo discreto y linealizado del sistema odométrico

La descripción no lineal y continua del modelo odométrico queda, por tanto, tal y como se muestra en la ecuación <2.22> y será necesario discretizarlo y luego linealizarlo para poder expresarlo con un formato como el mostrado en la ecuación <2.23>.

$$\begin{aligned}\dot{x}_o(t) &= V(t)\cdot\cos\theta_o(t)\\ \dot{y}_o(t) &= V(t)\cdot\sin\theta_o(t)\\ \dot{\theta}_o(t) &= \Omega(t)\end{aligned} \quad <2.24>$$

En primer lugar, como se desea utilizar el modelo para estimar la posición en un procesador digital, éste se discretiza, dando lugar a las siguientes expresiones:

$$\begin{aligned}x_{o,k} &= x_{o,k-1} + T_s\cdot V_k\cdot\cos\theta_{o,k-1}\\ y_{o,k} &= y_{o,k-1} + T_s\cdot V_k\cdot\sin\theta_{o,k-1}\\ \theta_{o,k} &= \theta_{o,k-1} + T_s\cdot\Omega_k\end{aligned} \quad <2.25>$$

Sin embargo, para poder aplicar el filtrado de Kalman al modelo es necesario conocer las matrices características del sistema de ecuaciones <2.23>. Por esto se aplica la descomposición en serie de Taylor a la ecuación de estado presentada en <2.24> y se obtiene el jacobiano que constituirá la representación lineal de las matrices A y B. A la hora de aplicar la linealización por Taylor se emplean únicamente el primer y segundo término de la serie, y se centran en el punto de trabajo del sistema[5].

$$f(\vec{x}_k) = f(\vec{x}_0) + A(\vec{x}_k - \vec{x}_0) + B(\vec{u}_k - \vec{u}_0),$$

donde

$$A \Rightarrow A_{[i,j]} = \frac{\partial f_{[i]}}{\partial \vec{x}_{[j]}}\left(\vec{x}_0, \vec{u}_0\right)$$

$$B \Rightarrow B_{[i,j]} = \frac{\partial f_{[i]}}{\partial \vec{u}_{[j]}}\left(\vec{x}_0, \vec{u}_0\right) \quad <2.26>[6]$$

$$A = \begin{bmatrix} \frac{\partial x_k}{\partial x_{k-1}}\left(x_0,\vec{u}_0\right) & \frac{\partial x_k}{\partial y_{k-1}}\left(y_0,\vec{u}_0\right) & \frac{\partial x_k}{\partial \theta_{k-1}}\left(\theta_0,\vec{u}_0\right)\\ \frac{\partial y_k}{\partial x_{k-1}}\left(x_0,\vec{u}_0\right) & \frac{\partial y_k}{\partial y_{k-1}}\left(y_0,\vec{u}_0\right) & \frac{\partial y_k}{\partial \theta_{k-1}}\left(\theta_0,\vec{u}_0\right)\\ \frac{\partial \theta_k}{\partial x_{k-1}}\left(x_0,\vec{u}_0\right) & \frac{\partial \theta_k}{\partial y_{k-1}}\left(y_0,\vec{u}_0\right) & \frac{\partial \theta_k}{\partial \theta_{k-1}}\left(\theta_0,\vec{u}_0\right) \end{bmatrix} = \begin{bmatrix} 1 & 0 & -V_0\cdot T_s\cdot\sin(\theta_0)\\ 0 & 1 & V_0\cdot T_s\cdot\cos(\theta_0)\\ 0 & 0 & 1 \end{bmatrix} \quad <2.27>$$

[5] A partir de este punto, para simplificar la nomenclatura, se elimina la notación m_o (subíndice 'o') indicadora de que se trata del modelo odométrico del sistema móvil. Se recuperará cuando se obtenga el modelo completo.

[6] Se utilizará el subíndice '0' para indicar que se trata del punto de linealización de la serie y no crear confusión con la notación temporal.

$$B = \begin{bmatrix} \frac{\partial x_k}{\partial V_k}\left(\vec{x}_0, V_0\right) & \frac{\partial x_k}{\partial \Omega_k}\left(\vec{x}_0, \Omega_0\right) \\ \frac{\partial y_k}{\partial V_k}\left(\vec{x}_0, V_0\right) & \frac{\partial y_k}{\partial \Omega_k}\left(\vec{x}_0, \Omega_0\right) \\ \frac{\partial \theta_k}{\partial V_k}\left(\vec{x}_0, V_0\right) & \frac{\partial \theta_k}{\partial \Omega_k}\left(\vec{x}_0, \Omega_0\right) \end{bmatrix} = \begin{bmatrix} T_s \cdot \cos(\theta_0) & 0 \\ T_s \cdot \sin(\theta_0) & 0 \\ 0 & T_s \end{bmatrix} \quad <2.28>$$

Con todo ello, el modelo final en formato matricial del modelo odometrico discretizado y linealizado queda:

$$\begin{bmatrix} x_k \\ y_k \\ \theta_k \end{bmatrix} = \begin{bmatrix} x_0 \\ y_0 \\ \theta_0 \end{bmatrix} + \begin{bmatrix} 1 & 0 & -V_0 \cdot T_s \cdot \sin(\theta_0) \\ 0 & 1 & V_0 \cdot T_s \cdot \cos(\theta_0) \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_{k-1} - x_0 \\ y_{k-1} - y_0 \\ \theta_{k-1} - \theta_0 \end{bmatrix} + \begin{bmatrix} T_s \cdot \cos(\theta_0) & 0 \\ T_s \cdot \sin(\theta_0) & 0 \\ 0 & T_s \end{bmatrix} \begin{bmatrix} (V_k - V_0)^* \\ \Omega_k - \Omega_0 \end{bmatrix} \quad <2.29>$$

Y en formato de sistema de ecuaciones resulta:

$$\begin{aligned} x_{o,k} &= x_{o,k-1} - V_0 \cdot T_s \cdot \sin(\theta_{o,0})(\theta_{o,k-1} - \theta_{o,0}) \\ y_{o,k} &= y_{o,k-1} + V_0 \cdot T_s \cdot \cos(\theta_{o,0})(\theta_{o,k-1} - \theta_{o,0}) \\ \theta_{o,k} &= \theta_{o,k-1} + T_s(\Omega_k - \Omega_0) \end{aligned} \quad <2.30>$$

Antes de comenzar el siguiente punto es necesario puntualizar algo acerca del uso de los diferentes modelos obtenidos. Teniendo en cuenta la aplicación final que se les va a dar (implementación de un EKF para fusionar medidas de posición), es fácil observar que el modelo no lineal y discreto será el utilizado para estimar el vector de estado en la etapa de predicción, mientras que las matrices del modelo discreto y linealizado (A_k y C_k, si bien esta última es constante e igual a la matriz identidad) serán utilizadas para implementar las ecuaciones de evolución del estimador de Kalman.

2.2.1.2. Modelo del ruido asociado al vector de estado

Tal y como se resumía en la introducción de esta sección, el modelo odométrico permitirá obtener la estimación del vector de estado del sistema robótico mediante el uso de un filtro de Kalman que elimine el ruido de medidas introducido por los sensores odométricos.

Es decir, el comportamiento dinámico del sistema, representado por la ecuación de estados $x_k=f(x_{k-1},u_k,0)$, se estima sin tener en cuenta el ruido de medidas que afecta al estado mediante las ecuaciones <2.25> anteriores. Sin embargo, para modelar el comportamiento completo del sistema ha de incluirse la caracterización del ruido de medida comentado.

Realmente, como lo que se desea es utilizar la representación del ruido para aplicarla al algoritmo del EKF, es necesario linealizarla y presentarla de forma discreta del mismo modo que se hizo con el modelo odométrico sin ruido

en el apartado anterior para llegar, al final de la caracterización, a una expresión del tipo:

$$f(\vec{x}_k) = \vec{x}_0 + A(\vec{x}_k - \vec{x}_0) + ... + W_k \cdot \vec{w}_k \quad <2.31>$$

Donde $W_k \Rightarrow W_{[i,j]} = \frac{\partial f_{[i]}}{\partial \vec{w}_{[j]}}(\vec{x}_0, \vec{u}_0)$ es la matriz que asocia el ruido asociado a la medida con el vector de estados.

El planteamiento parte, por tanto, del análisis del punto en el que se incorporan los ruidos de medida de los encoders:

$$\begin{aligned} \omega_{In} &= \omega_I + w_I \\ \omega_{Dn} &= \omega_D + w_D \end{aligned} \quad <2.32>$$

Aplicando esa nueva definición a la cinemática inversa del robot (ver ecuación <2.22>) resulta lo siguiente:

$$\begin{aligned} V_n &= \frac{R}{2}(\omega_{Dn} + \omega_{In}) = V + \frac{R}{2}(w_D + w_I) \\ \Omega_n &= \frac{R}{D}(\omega_{Dn} - \omega_{In}) = \Omega + \frac{R}{D}(w_D + w_I) \end{aligned} \quad <2.33>$$

Con esto, el modelo no lineal discreto resultante queda como sigue (de la ecuación <2.25>):

$$\begin{aligned} x_{on,k} &= x_{o,k-1} + T_s \cdot V_{n,k} \cdot \cos\theta_{o,k-1} = x_{o,k-1} + T_s \cdot V_k \cdot \cos\theta_{o,k-1} + T_s \cdot \cos\theta_{o,k-1} \cdot \frac{R}{2}(w_D + w_I) = x_{o,k} + T_s \cdot \cos\theta_{o,k-1} \cdot \frac{R}{2}(w_D + w_I) \\ y_{on,k} &= y_{o,k-1} + T_s \cdot V_{n,k} \cdot \sin\theta_{o,k-1} = y_{o,k-1} + T_s \cdot V_k \cdot \sin\theta_{o,k-1} + T_s \cdot \sin\theta_{o,k-1} \cdot \frac{R}{2}(w_D + w_I) = y_{o,k} + T_s \cdot \sin\theta_{o,k-1} \cdot \frac{R}{2}(w_D + w_I) \\ \theta_{on,k} &= \theta_{o,k-1} + T_s \cdot \Omega_{n,k} = \theta_{o,k-1} + T_s \cdot \Omega_k + T_s \cdot \frac{R}{D}(w_D - w_I) = \theta_{o,k} + T_s \cdot \frac{R}{D}(w_D - w_I) \end{aligned}$$

<2.34>

A este modelo se le aplica la linealización que se presentó en la ecuación <2.31>. Se desea calcular la relación lineal entre el vector ruido $\vec{w}_k = \begin{bmatrix} w_D & w_I \end{bmatrix}^T$ y el vector de estado $\vec{x}_k = \begin{bmatrix} x_{o,k} & y_{o,k} & \theta_{o,k} \end{bmatrix}^T$ obteniéndose los siguientes resultados:

$$W_k = \begin{bmatrix} \frac{\partial x_k}{\partial w_D}(\vec{x}_0, \vec{u}_0) & \frac{\partial x_k}{\partial w_I}(\vec{x}_0, \vec{u}_0) \\ \frac{\partial y_k}{\partial w_D}(\vec{x}_0, \vec{u}_0) & \frac{\partial y_k}{\partial w_I}(\vec{x}_0, \vec{u}_0) \\ \frac{\partial \theta_k}{\partial w_D}(\vec{x}_0, \vec{u}_0) & \frac{\partial \theta_k}{\partial w_I}(\vec{x}_0, \vec{u}_0) \end{bmatrix} = \begin{bmatrix} R/2 \cdot T_s \cdot \cos(\theta_0) & R/2 \cdot T_s \cdot \cos(\theta_0) \\ R/2 \cdot T_s \cdot \sin(\theta_0) & R/2 \cdot T_s \cdot \sin(\theta_0) \\ R/D \cdot T_s & -R/D \cdot T_s \end{bmatrix} \quad <2.35>$$

Con esta matriz y las obtenidas en el apartado anterior (A_K y B_K) se puede presentar un modelo completo discreto y linealizado con ruido del sistema odométrico. Sin embargo, teniendo en cuenta lo explicado en el desarrollo del EKF en la sección anterior, queda claro que el ruido no se incluye en el modelo a utilizar en el filtrado sino en la obtención de la matriz de innovación del covarianza del error de estimación ($P_{k+1/k}$), que se calcula de la siguiente forma:

$$P_{k+1/k} = A_k \cdot P_{k/k} \cdot A_k^T + W_k \cdot Q \cdot W_k^T \quad <2.36>$$

A la vista de esta ecuación se observa que se han obtenido ya todos los parámetros necesarios para implementar el filtrado de Kalman sobre el modelo odométrico, a falta únicamente, de la matriz de covarianza del ruido del sistema Q y que tendrá un valor:

$$Q = E\left[\vec{w}_k \cdot \vec{w}_k^T\right] = \begin{bmatrix} \sigma^2_{w_D} & 0 \\ 0 & \sigma^2_{w_I} \end{bmatrix} \quad <2.37>$$

El valor de esta matriz no se ha calculado en este trabajo pero se obtiene, tal y como se explicaba en la sección anterior, mediante pruebas off-line que permitan realizar una estadística sobre el comportamiento ruidoso de las medidas de odometria, con lo que el modelo de ruido quedaría:

$$f(\vec{x}_k) = f(\vec{x}_0) + A(\vec{x}_k - \vec{x}_0) + \ldots + W_k \cdot \vec{w}_k \text{, donde}$$

$$W_k \cdot \vec{w}_k = \begin{bmatrix} R/2 \cdot T_s \cdot \cos(\theta_0) & R/2 \cdot T_s \cdot \cos(\theta_0) \\ R/2 \cdot T_s \cdot \sin(\theta_0) & R/2 \cdot T_s \cdot \sin(\theta_0) \\ R/D \cdot T_s & -R/D \cdot T_s \end{bmatrix} \cdot \begin{bmatrix} w_D \\ w_I \end{bmatrix}$$

Estando el vector de ruido $\vec{w}_k$ definido por sus estadísticos: covarianza $Q = \begin{bmatrix} \sigma^2_{w_D} & 0 \\ 0 & \sigma^2_{w_I} \end{bmatrix}$ y media nula.

Se pueden obtener otros modelos de este ruido a partir de los diferentes trabajos que han sido realizados a propósito del tema del ruido producido por los modelos de posicionamiento por dead-reckoning. Una descripción de algunos de ellos se encuentra en [Marrón-03].

2.2.2. Modelo del sistema de visión

Como ya se comentó, el segundo de los elementos que proporcionan información sobre la posición del robot será un sistema de posicionamiento por visión basado en el reconocimiento y caracterización de marcas artificiales en el entorno (ver figura 2.4). Será necesario obtener un modelo de este segundo

sistema para poder fusionar la información que proporcione con la suministrada por el modelo odométrico.

El sistema de posicionamiento local (SPL), diseñado por el Dr. D. Juan Carlos García García en su tesis doctoral [García-01], se basa en la detección y procesamiento por visión artificial de unas marcas artificiales especiales que se encuentran localizadas estratégicamente en el entorno de movimiento del robot. Los algoritmos desarrollados en su trabajo permiten, una vez localizada la marca en la imagen del entorno, obtener la posición relativa de la cámara con respecto a la marca y, por lo tanto, del móvil ya que la anterior va solidaria a éste. En la figura 2.7 se muestra dicha relación (vector V1 en la figura), así como un gráfico esquematizado de la marca artificial específicamente diseñada para el algoritmo SPL.

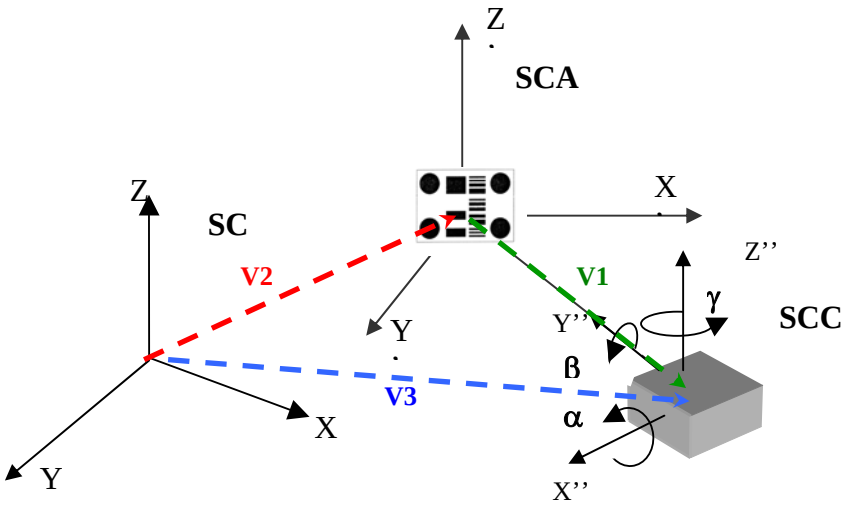


Figura.2.7. Representación espacial de los sistemas de coordenadas global del entorno (SCG XYZ), local de la marca (SCA, X'Y'Z') y de la cámara (SCC X''Y''Z''). Vectores de relación entre los tres sistemas:
SCC-SCA en verde
SCG-SCA en rojo
SCG-SCC en azul

Tal y como se mostraba en la figura 2.4, la salida del algoritmo SPL se utiliza para obtener la posición absoluta del robot en el entorno de movimiento ($\vec{z}_{v,k}$) mediante una transformación simple. Para poder realizar esta transformación el algoritmo SPL prevée que la marca artificial diseñada al efecto incluye un código de barras que la identifica unívocamente dentro del entorno de movimiento y, a través de él, se obtiene su posición absoluta en el entorno (vector V2 en la figura 2.7) y, con ella, la del robot (vector V3 en la figura 2.7), gracias a un mapa del medio generado off-line que ha de poseer el sistema de visión para realizar el posicionamiento absoluto. La figura 2.8 muestra este mecanismo.

El sistema de navegación realizado en el trabajo [Marrón-00] utiliza estas marcas artificiales como nodos en la planificación de rutas y de trayectorias.

Tanto es así que el mapa comentado no sólo incluye información acerca de la posición absoluta de cada marca sino también ciertos datos interesantes para el software de navegación a la hora de diseñar las trayectorias que ha de realizar el robot.

El papel del modelo de visión en la implementación del EKF se reduce a proporcionar el valor del vector de salida de posición sensada en cada momento ($\vec{z}_{v,k} = \begin{bmatrix} x_{v,k} & y_{v,k} & \theta_{v,k} \end{bmatrix}^T = \vec{x}_{v,k}$). Esta información coincide totalmente con la salida de posición global del algoritmo de visión ya comentada, por lo que no será necesario implementar ningún modelo específico de este sistema para incorporar la información sensada al EKF.

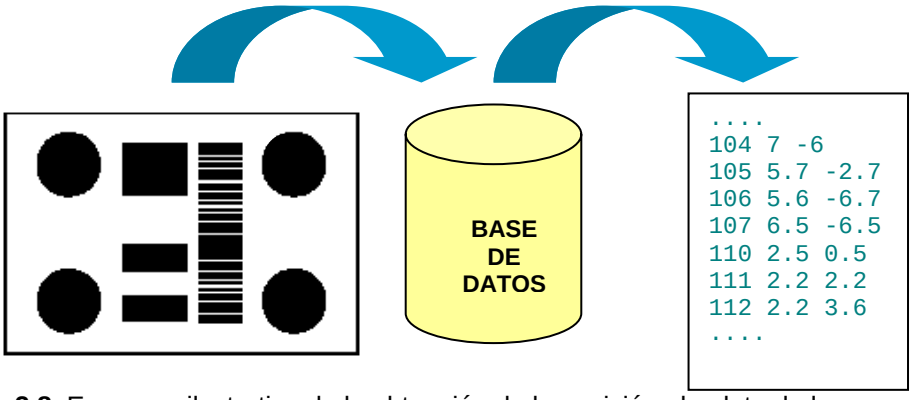


Figura 2.8. Esquema ilustrativo de la obtención de la posición absoluta de la marca en el SCG mediante el código de barras

El sistema de visión constituirá, por tanto, el elemento sensorial que proporciona el vector de salida en el algoritmo de fusión que se presentó en la descripción del EKF a través de la siguiente ecuación:

$$\vec{z}_k = f\left(\vec{x}_k, \vec{v}_k\right), \quad \text{o directamente} \quad \vec{z}_k = \vec{x}_k + V_k \cdot \vec{v}_k \qquad <2.38>^7$$

Además, tal y como se observa en la ecuación anterior, será necesario cuantificar mediante la consiguiente matriz de covarianza (R) y de relación (V_k) el ruido asociado a esta nueva medida.

2.2.2.1. Modelo del ruido asociado al vector de salida

En la tesis [García-01] se desarrolla un algoritmo SPL simplificado realizando varias suposiciones previas que se detallarán más adelante. Esta versión simplificada será la que se use en este trabajo. En la tesis se realiza, además, un estudio detallado para analizar los errores que incorporan los resultados de posición debido a circunstancias diversas: iluminación ambiente, distancia móvil-marca, orientación de la marca, etc. Todas estas causas se

[7] Se ha planteado esta simplificación teniendo en cuenta que la salida coincide con el vector de estados en este sistema y que el ruido y la salida se hacen independientes a través de la correspondiente matriz de asociación V_k, que no ha de confundirse con la entrada de velocidad del modelo odométrico.

agrupan y se caracterizan por lo que se ha dado en llamar "varianza del error de píxel".

Esta varianza se extrae del error obtenido en el cálculo de los centroides de los círculos que aparecen en la marca artificial, y que son la base para obtener la posición local relativa del robot.

Para llevar a cabo el análisis, el autor de la tesis realizó experimentos con un banco de pruebas de 300 imágenes capturadas en 30 puntos diferentes del entorno de la marca (10 imágenes por punto de prueba), organizados a distintas distancias (de 1 a 5m con incrementos de 1m entre puntos de prueba) y orientaciones (de 0 a -75º de orientación del eje de proyección de la cámara con respecto a la marca, con incrementos entre puntos de prueba de -15º). Las imágenes se capturaron con características muy distintas de iluminación y con un fondo no preparado, por lo que representan fielmente la situación del algoritmo SPL en funcionamiento normal.

Con estas condiciones se procesan todas las imágenes con el algoritmo simplificado y se obtiene un error de píxel de varianza muy semejante en todos los puntos de prueba y de un valor de alrededor de los 0.011 pixel^2.

Además el autor también realiza un estudio sobre la propagación de este ruido a la salida de posición de su algoritmo ($\vec{z}'_{v,k}$), lo cual es de gran utilidad en este trabajo para incorporar la información de ruido al EKF. Concretamente, el análisis permite obtener la matriz de covarianza R asociada a un vector transformado (en coordenadas polares) que va a permitir analizar de forma más gráfica las elipses de error del algoritmo SPL. El vector utilizado para caracterizar el ruido es el siguiente:

$$\vec{z}''_{v,k} = \begin{bmatrix} r_k & \sin(\gamma_k) & \cos(\gamma_k) \end{bmatrix}^T \quad <2.39>$$

Donde r_k es el promediado de distancia del robot a la marca, sobre el eje óptico de la cámara (SCC). Para poder relacionar este vector $\vec{z}''_{v,k}$ con el vector $\vec{z}'_{v,k}$ es necesario usar las ecuaciones de transformación tridimensional correspondientes, si bien aplicando las condiciones expuestas en el método simplificado (β=0 y α conocido) la relación no lineal existente entre los dos vectores puede obtenerse mediante sencillas transformaciones tridimensionales, tal y como se muestra en la figura 2.9.

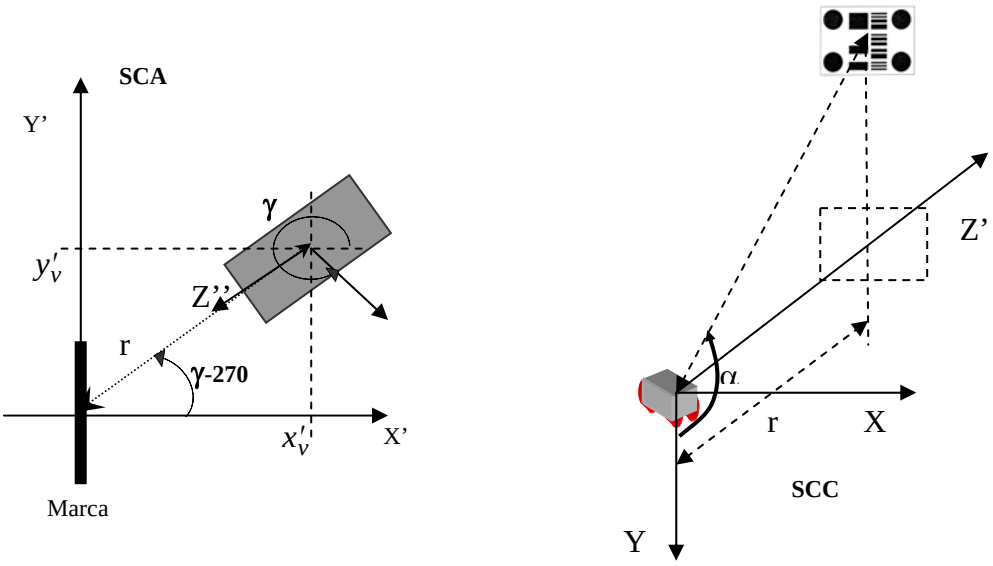


Figura 2.9. Transformación inversa del vector de coordenadas polares de caracterización del ruido en el SPL

A partir de la figura es fácil obtener las relaciones buscadas, que vendrán dadas por las expresiones que se muestra a continuación:

$$\begin{aligned} x'_{v,k} &= r_k \cdot \cos(\gamma_k - 270) = -r_k \cdot \sin(\gamma_k) \\ y'_{v,k} &= r_k \cdot \sin(\gamma_k - 270) = r_k \cdot \cos(\gamma_k) \\ \theta'_{v,k} &= \gamma_k \end{aligned} \quad <2.40>$$

Además es necesario aplicar a estos resultados la transformación SCC-SCG para relacionar finalmente el vector de ruido $\vec{v}_k = \begin{bmatrix} v_r & v_{\sin\gamma} & v_{\cos\gamma} \end{bmatrix}^T$ con el de salida de posición $\vec{z}_{v,k}$:

$$\begin{aligned} x_{v,k} &= x_{m,i} + x'_{v,k} \cdot \cos(\theta_{m,i}) - y'_{v,k} \cdot \sin(\theta_{m,i}) = x_{m,i} - r_k \cdot \sin(\gamma_k) \cdot \cos(\theta_{m,i}) - r_k \cdot \cos(\gamma_k) \cdot \sin(\theta_{m,i}) \\ y_{v,k} &= y_{m,i} + x'_{v,k} \cdot \sin(\theta_{m,i}) + y'_{v,k} \cdot \cos(\theta_{m,i}) = y_{m,i} + r_k \cdot \sin(\gamma_k) \cdot \sin(\theta_{m,i}) + r_k \cdot \cos(\gamma_k) \cdot \cos(\theta_{m,i}) \\ \theta_{v,k} &= \theta_{m,i} + \theta'_{v,k} = \theta_{m,i} + \gamma_k \end{aligned}$$

<2.41>

Hay que tener en cuenta que en el anterior sistema de ecuaciones, el vector de posición absoluta de la marca artificial $P_{m,i} = \begin{bmatrix} x_{m,i} & y_{m,i} & \theta_{m,i} \end{bmatrix}^T$ es conocido, por lo que realmente sólo hay dos variables: r_k y γ_k. El vector $P_{m,i}$ se corresponde con el vector V2 en la figura 2.7. En él, $x_{m,i}$ e $y_{m,i}$ expresan la posición absoluta de la marca y $\theta_{m,i}$ da una idea de la orientación de la perpendicular a la marca en el sistema de coordenadas absoluto.

El problema es que la expresión anterior (<2.41>) no tiene el formato típico de modelado del vector de salida presentado desde el principio del

estudio ($\vec{z}_{v,k} = h\left(\vec{x}_{v,k}, \vec{v}_k\right)$), pues éste no muestra ninguna relación con el vector de estado ($\vec{x}_{v,k}$). El hecho es que, tal y como se planteó en los comentarios iniciales de este apartado, no se desea obtener un modelo específico del sistema de visión sino que la salida $\vec{z}'_{v,k}$ del algoritmo SPL será preprocesada antes de ser incorporada al EKF con el fin de obtener $\vec{z}_{v,k}$.

Sin embargo, el algoritmo EKF sí requiere de una matriz de covarianza R_K que identifique al ruido de sensado del vector de salida ($\vec{z}_{v,k}$) y, si la matriz de covarianza no está relacionada de forma lineal con este vector de salida o, como ocurre en esta aplicación, ni tan siquiera está relacionada con dicho vector de salida, será necesario buscar la matriz V_k que relacione de forma lineal el vector de ruido $\vec{v}''_k = \begin{bmatrix} v_r & v_{\sin\gamma} & v_{\sin\gamma} \end{bmatrix}^T$, al que define la matriz R, con el vector de salida $\vec{z}_{v,k}$.

El objetivo es idéntico al presentado a la hora de obtener el modelo del ruido de odometría: poder obtener la ecuación de salida (que procede en este caso del modelo de visión) de forma lineal, para poder aplicar así la definición del EKF tal y como se presentaba en la ecuación <2.12> y que aquí se repite:

$$h(\vec{x}_k) = h(\vec{x}_0) + C \cdot \left(\vec{x}_k - \vec{x}_0\right) + V_k \cdot \vec{v}_k = \vec{z}_k + V_k \cdot \vec{v}_k \quad <2.42>$$

En dicha ecuación se ha eliminado la relación de modelado entre el vector de salida y el de estado (que no se requiere, tal y como ya se ha comentado), suponiendo entonces que $\vec{x}_{v,k} = \vec{z}_{v,k}$, y que por tanto C es una matriz identidad. Además, la matriz de relación entre el vector de salida y el de ruido se calcula como: $V_k \Rightarrow V_{[i,j]} = \frac{\partial h_{[i]}}{\partial v_{[j]}}\left(\vec{x}_k, 0\right)$, tal y como se hizo con W_k en el apartado anterior. Finalmente, el vector de ruido de medida de la salida de posición será en este caso el ya mencionado, de componentes $\vec{v}''_k = \begin{bmatrix} v_r & v_{\sin\gamma} & v_{\sin\gamma} \end{bmatrix}^T$.

Con todo ello podemos simplificar la expresión <4.41> del siguiente modo:

$$\begin{aligned} x_{vn,k} &= cte1 - (r_k + v_r) \cdot (\sin(\gamma_k) + v_{\sin\gamma}) \cdot cte2 - (r_k + v_r) \cdot (\cos(\gamma_k) + v_{\cos\gamma}) \cdot cte3 \\ y_{vn,k} &= cte4 + (r_k + v_r) \cdot (\sin(\gamma_k) + v_{\sin\gamma}) \cdot cte3 + (r_k + v_r) \cdot (\cos(\gamma_k) + v_{\cos\gamma}) \cdot cte2 \quad <2.43> \\ \theta_{vn,k} &= cte5 + \gamma_k \end{aligned}$$

Donde $P_{m,i} = \begin{bmatrix} x_{m,i} & y_{m,i} & \theta_{m,i} \end{bmatrix}^T = \begin{bmatrix} cte1 & cte4 & cte5 \end{bmatrix}^T$, $cte2 = \cos(\theta_{m,i})$ y $cte3 = \sin(\theta_{m,i})$.

Aplicando entonces la definición de la matriz V_k para completar el modelo buscado, se obtiene lo siguiente:

$$V_k = \begin{bmatrix} \frac{\partial x_k}{\partial v_r}\left(\vec{x}_0\right) & \frac{\partial x_k}{\partial v_{\sin\gamma}}\left(\vec{x}_0\right) & \frac{\partial x_k}{\partial v_{\cos\gamma}}\left(\vec{x}_0\right) \\ \frac{\partial y_k}{\partial v_r}\left(\vec{x}_0\right) & \frac{\partial y_k}{\partial v_{\sin\gamma}}\left(\vec{x}_0\right) & \frac{\partial y_k}{\partial v_{\cos\gamma}}\left(\vec{x}_0\right) \\ \frac{\partial \theta_k}{\partial v_r}\left(\vec{x}_0\right) & \frac{\partial \theta_k}{\partial v_{\sin\gamma}}\left(\vec{x}_0\right) & \frac{\partial \theta_k}{\partial v_{\cos\gamma}}\left(\vec{x}_0\right) \end{bmatrix} = \begin{bmatrix} -\sin(\gamma_k)\cdot cte2 & \sin(\gamma_k)\cdot cte3 & 0 \\ -r_k\cdot cte2 & r_k\cdot cte2 & 0 \\ -r_k\cdot cte3 & r_k\cdot cte2 & 0 \end{bmatrix} \quad <2.44>$$

Una vez obtenida la matriz que relaciona $\vec{v}_k''$ con $\vec{z}_{v,k}$ solamente falta presentar la matriz de covarianza del ruido de salida R_K, que, tal y como ya se ha comentado se define perfectamente en la tesis doctoral que sirve de base de este trabajo.

En el trabajo desarrollado en la tesis se ha asegurado que la variable de ruido de medida $\vec{v}_k''$ asociado al vector de salida, tiene media nula gracias al proceso de calibración de la cámara. Su efecto se observa claramente en la figura 2.10, donde se muestran las elipses de error del algoritmo de posicionamiento para el banco de imágenes ya comentado. En esta figura se observa claramente que todas las elipses de medida con error están centradas en el punto de prueba correspondiente.

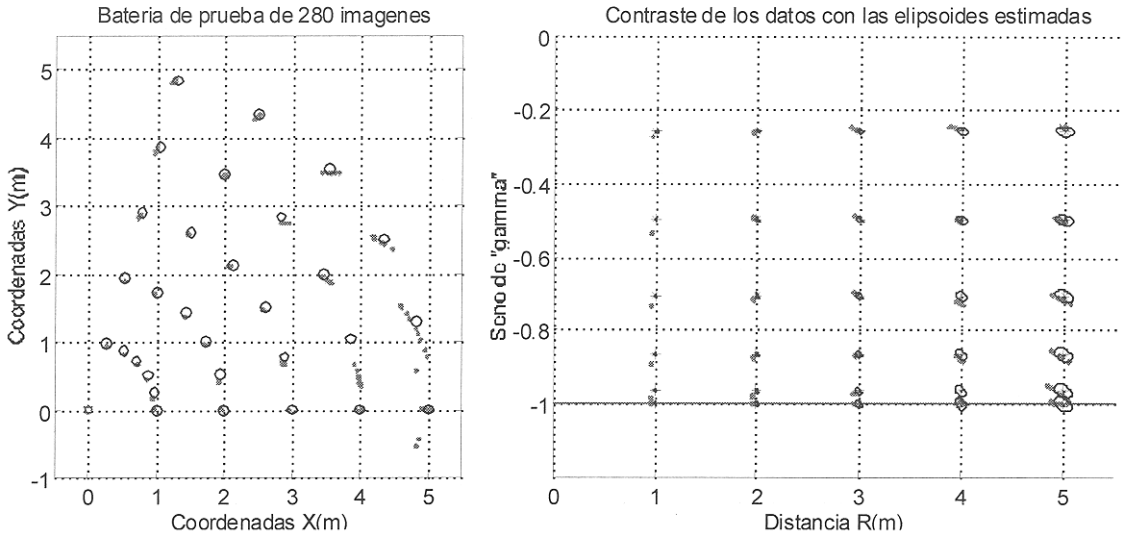


Figura 2.10. Resultado del ruido de medidas del sistema de visión con la batería de imágenes de prueba

La matriz de covarianza de ruido de salida (R) obtenida con el sistema de visión, tendrá por tanto la siguiente forma:

$$R_k = E\left[\vec{v}_k \cdot \vec{v}_k^{\,T}\right] = \begin{bmatrix} \sigma_r^2 & 0 & 0 \\ 0 & \sigma_{\sin\gamma}^2 & 0 \\ 0 & 0 & \sigma_{\cos\gamma}^2 \end{bmatrix} \quad <2.45>$$

En la tesis [García-01] se presenta el modo de calcular el valor instantáneo de esta matriz. Este cálculo no ha sido incluido en este documento pues incluye un proceso complejo de cálculo matricial que debe realizarse en función de muchas de las variables que entran en juego en el algoritmo SPL: el vector de orientación de la cámara con respecto a la marca, la posición de los centroides dentro de la imagen e incluso la distancia focal de la cámara, y evidentemente la covarianza del error de pixel. Va a ser, por tanto, una matriz de covarianza dinámica.

Así se puede completar el modelo del ruido de salida, es decir de la medida de posición obtenido con el sistema de visión, con la siguiente expresión:

$$h(\vec{x}_k) = h(\vec{x}_0) + C(\vec{x}_k - \vec{x}_0) + \ldots + V_k \cdot \vec{v}_k \text{ ,con}$$

$$V_k \cdot \vec{v}_k = \begin{bmatrix} -\sin(\gamma_k)\cdot cte2 & \sin(\gamma_k)\cdot cte3 & 0 \\ -r_k \cdot cte2 & r_k \cdot cte2 & 0 \\ -r_k \cdot cte3 & r_k \cdot cte2 & 0 \end{bmatrix} \cdot \begin{bmatrix} v_r \\ v_{\sin\gamma} \\ v_{\cos\gamma} \end{bmatrix}$$

<2.46>

estando el vector $\vec{v}_k$ definido por sus estadísticos de media nula y covarianza:

$$R_k = \begin{bmatrix} \sigma_r^2 & 0 & 0 \\ 0 & \sigma_{\sin\gamma}^2 & 0 \\ 0 & 0 & \sigma_{\cos\gamma}^2 \end{bmatrix}$$

3. Trabajos Previos

En este capítulo se pasan a exponer los trabajos previos en los que está basado este proyecto. Si bien ya se han comentado prácticamente en su totalidad sus fundamentos teóricos, ahora se pasan a presentar las implementaciónes reales de estos conceptos de las cuales se ha partido para dar forma a la aplicación aquí presentada.

En primer lugar, se describen los trabajos realizados en [Marrón-00] y que sirven como base principal de este proyecto. En este trabajo de fin de carrera se implementó la aplicación de navegación y control que pasa a ser, en la aplicación descrita en la presente memoria, el proceso principal.

Después se comenta el trabajo descrito en [Lopez-02], dirigido a implementar el algoritmo de posicionamiento diseñado en [García-00], que ya ha sido explicado en el capítulo anterior.

Por último, se describe la primera aproximación al objetivo de este proyecto, realizada en [Marrón-02] y consistente en una simulación parcial de la funcionalidad aquí implementada empleando Matlab y sus herramientas para tiempo real (Real Time Workshop).

3.1 Navegación

El primero de los trabajos en los que se basa el presente proyecto constituyó el trabajo de fin de carrera de Dña. Marta Marrón Romera, que está documentado en [Marrón-00]. En este proyecto se propuso una solución robusta para el modelado de entornos cerrados en los que se encuentra, al menos, cierta estructuración y se implementó un sistema de navegación que permitía a un móvil desplazarse de forma autónoma de un lugar a otro dentro

de uno de esos entornos estructurados (la Escuela Politécnica de la Universidad de Alcalá).

Además, se presentó como característica el hecho de que el propio edificio contuviera la información necesaria para desarrollar el movimiento, lo que permitía extrapolar el problema a casi cualquier otro entorno. En cuanto a la generación de trayectorias, se basaba en simular los movimientos que desarrollaría un humano para moverse en este tipo de entornos: curvas simples y líneas rectas, de modo que el movimiento resultara confortable para el usuario.

Los procesos básicos de este sistema de navegación consistían en:

- ✓ El **gestor de procesos**, encargado de **sincronizar** el resto de procesos y supervisar la ejecución. A través de él, se realizaban las llamadas a los procesos necesarios, se gestionaban las comunicaciones entre ellos, etc.

- ✓ El **planificador de rutas**, que obtenía el camino más corto entre el origen y el destino indicados por el usuario. Para llevar a cabo esta tarea se empleaban algoritmos basados en diagramas de nodos y se usó la organización jerárquica exixtente en el mapa.

- ✓ El **generador de trayectorias**, que diseñaba la trayectoria que habría de seguir el móvil a lo largo del camino.

- ✓ El **controlador de posición**, que se encargaba de transformar las consignas de posición en actuaciones de velocidad para enviar a las ruedas del robot, y de conseguir que la trayectoria se siguiera con una determinada dinámica de control.

Aparte de los procesos básicos, fueron necesarios otros elementos para que el movimiento se llevara a cabo. Estos eran:

- ✓ El **sistema de bajo nivel**, con los motores controlados con lazos independientes y sus tarjetas excitadoras.

- ✓ La **interfaz de usuario**, a través del cual se introducía el objetivo del proceso de navegación.

- ✓ Un **mapa del entorno**, necesario para poder desarrollar la tarea de navegación.

- ✓ Un **sistema de posicionamiento**, que permitía al controlador de posición conocer la posición real del robot para compararla con la consigna y actuar en consecuencia sobre los motores del móvil.

Un esquema de la aplicación sería el siguiente:

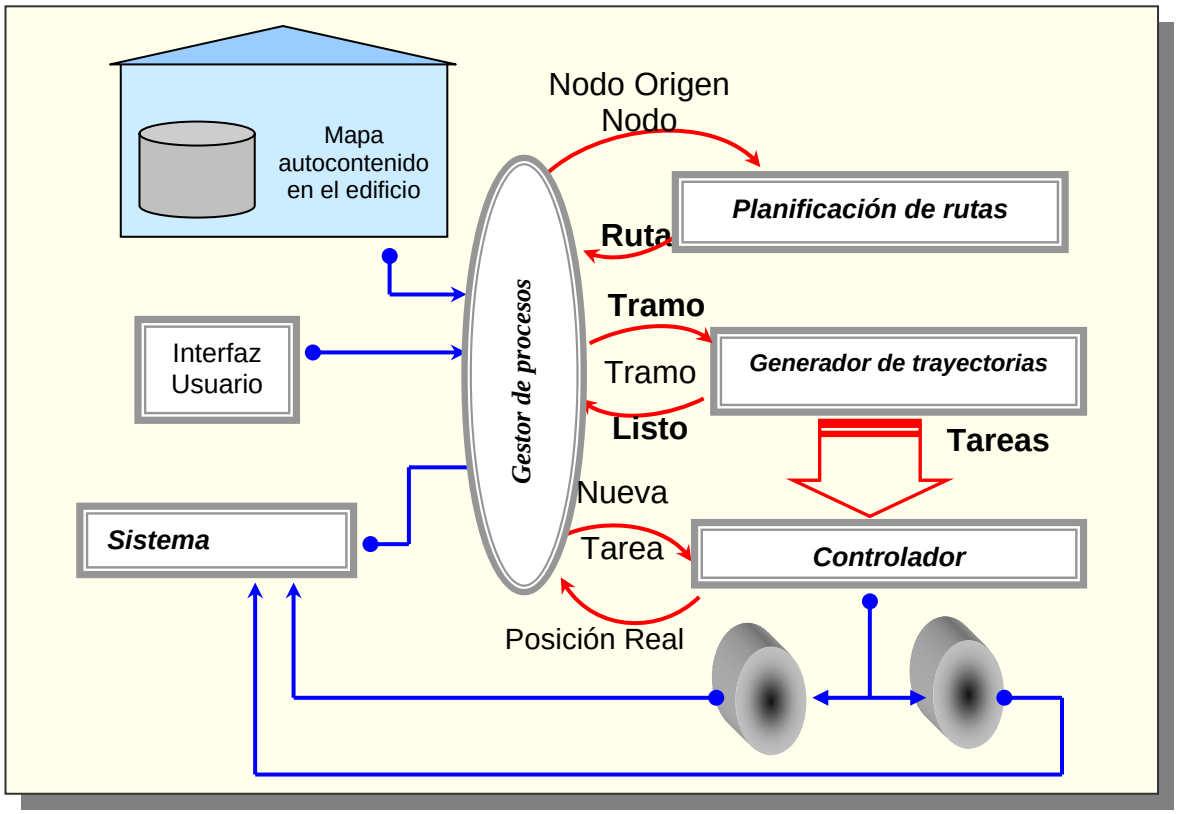


Figura 3.1. Diagrama funcional del sistema de navegación.

3.1.1. Descripción general de los procesos básicos de navegación.

En este apartado se pretende realizar una pequeña descripción del funcionamiento de cada una de las funciones referidas arriba.

Gestor de Procesos.

Como ya se ha comentado, el gestor de procesos tenía la función de organizar y supervisar de las tareas que desarrollaba el sistema de navegación global. Así, en un ejemplo típico de movimiento de la silla de ruedas entre un punto de origen y otro de destino, el gestor de procesos realizaría las siguientes funciones:

1. Recoger de la interfaz de usuario las consignas de punto de inicio y fin del movimiento.
2. Realizar la llamada al planificador de caminos y recoger la ruta[8] que proporciona como resultado.
3. Dividir la ruta en tramos[9] y realizar las llamadas pertinentes al generador de trayectorias para obtener las tareas correspondientes a cada tramo.

[8] Conjunto de nodos a lo largo del recorrido entre el nodo origen y el destino.

[9] En este contexto se debe entender tramo como el recorrido entre dos nodos adyacentes de la ruta.

4. Supervisar la ejecución del algoritmo de control cada periodo de muestreo, así como el movimiento real del robot.

Planificador de Rutas.

De la primera tarea de la aplicación de navegación se encargaba el Planificador de Rutas. Este elemento recibía las consignas directamente de la interfaz de usuario.

El hecho de trabajar mediante diagramas de nodos, unido al modelado jerárquico del entorno que se ha desarrollado, facilitaba mucho este proceso, tal y como se irá viendo a lo largo del trabajo. Este modelado se explica en el apartado 3.1.3.

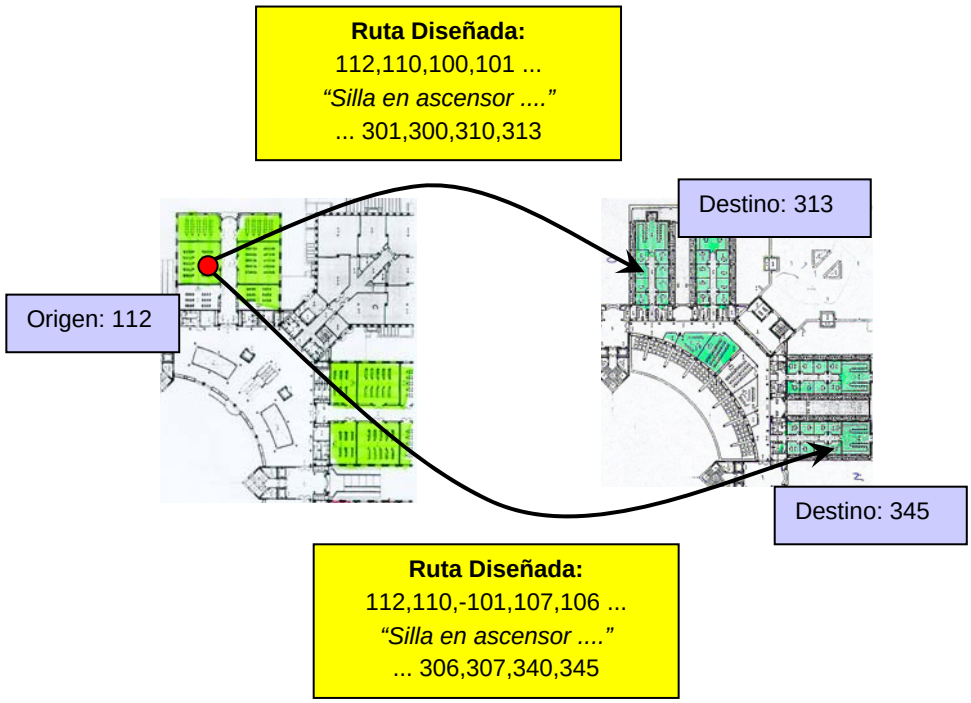


Figura 3.2. Ejemplo de diseño de Ruta.

El Planificador de Rutas necesitaba información del entorno para trazar la ruta que une el destino con la posición del móvil en ese momento. Esa información era proporcionada por el mapa del entorno, que presentaba una distribución jerárquica en forma de nodos, y estaba autocontenida en el edificio. Además, el planificador de rutas había de estar comunicado con el módulo de interfaz de usuario, cosa que ocurría indirectamente a través del gestor de procesos.

Generador de Trayectorias.

Este elemento recibía del gestor de procesos un tramo que tenía que subdividir en tareas y obtener para cada una de ellas las consignas que necesitaba el controlador de posición para gestionar la ejecución de la trayectoria asociada a cada una de las tareas. Para unir los dos nodos que

conforman un tramo, el generador diseñaba 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.

Para realizar su cometido, el generador de trayectorias tenía acceso directo al mapa del entorno, lo cual le permitía determinar en qué casos era necesario incorporar una tarea de avance y en cuales una de giro.

Figura 3.3. Ejemplo de división en tareas de un trayecto concreto.

Los resultados obtenidos de la generación de trayectorias tenían que pasar al controlador, para lo cual se empleaba un sistema de buffers controlados por flags de sincronismo que controlaba el gestor de procesos.

Controlador de Posición.

Se trataba de un controlador de posición que, comparando la posición real del móvil (obtenida a partir de un proceso de dead reckoning) y del tipo de trayectoria que se deseaba describir, obtenía las consignas de bajo nivel de velocidad angular que debían seguir las ruedas del robot.

Para poder llevar a cabo el control, este último módulo requería de un sistema de realimentación de la posición, que le permitiese comparar la posición consigna con la real del móvil y obtener a partir del error de posición las consignas de velocidad. En este proyecto se empleó 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.

3.1.2. El bajo nivel

Evidentemente, para poder desarrollar los algoritmos de navegación fue necesario emplear una silla de ruedas perfectamente acondicionada, como la mostrada en la figura 3.4.

Figura 3.4. La Silla de Ruedas Autónoma empleada en la aplicación

Para ello se empleó una plataforma que contaba con dos motores de continua excitados por sendos puentes en H, que, a su vez, eran controlados por microcontroladores programados al efecto. Todo el desarrollo del bajo nivel se llevó a cabo en otros proyectos como, por ejemplo, [Sebastián-99].

Como controladores de bajo nivel se empleaba el integrado Neuron Chip 3120 de Echelon, sistema orientado a la implementación de redes de control distribuido. Empleando esta característica, la plataforma constaba de una red de control distribuido basado en el protocolo LonWorks (EIA709), desarrollado por Echelon que implementa la torre de protocolos de comunicación OSI completa en el propio Neuron Chip, haciendo el trabajo de gestión de la comunicación a través de la red de control transparente al usuario.

Los integrados de Echelon implementan, además, sobre los motores de continua de la silla un control PI de velocidad que permitía regular con error nulo en régimen permanente la consigna de velocidad enviada por el navegador de alto nivel.

Para implementar la comunicación entre el navegador y el bajo nivel se empleó una tarjeta de interfaz que permitía 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 el trabajo previo [Persson-99], contaba con una memoria Dual Port que permitía comunicar los dos elementos consiguiendo una velocidad de transferencia de alrededor de 1Mbps. Es necesario comentar que para la nueva aplicación diseñada en este trabajo se ha prescindido de dicha tarjeta, como se explicará más adelante.

Se puede encontrar una descripción del protocolo y las comunicaciones el el Anexo A de esta memoria.

3.1.3. El modelado del entorno

El modelado del entorno en que se va a mover la silla de ruedas es imprescindible para desarrollar los procesos de navegación. En el proyecto tratado en esta memoria se utiliza exactamente el mismo modelo que se implementó para el proyecto [Marta-00], por lo que se puede encontrar una discusión extensiva sobre el tema en su documentación. Aquí explicaremos los conceptos más importantes de este diseño.

El modelado consiste en la interpretación del entorno de movimiento mediante un grafo que constituirá el mapa del entorno, y es una de las partes más importantes en los algoritmos de navegación, ya que permitirá al resto de los procesos de la aplicación tener la información que necesitan para gestionar el movimiento.

El diseño del mapa se adapta al medio de navegación de la aplicación: 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 se desarrolló 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.

Una gran cantidad de métodos de modelado de entorno se basan en la reducción del mapa a algún tipo de grafo sobre el que se aplicará un algoritmo de planificación de caminos. En este proyecto se emplea una técnica de tipo proyectivo, esto es, consistente en dividir el espacio de configuración en celdas, representando cada una de ellas un nodo.

Los nodos del mapa se asignan según las premisas siguientes:

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

- ✓ Los nodos constituyen además un grafo jerárquico 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 cualesquiera será mediante el mapa jerárquico de nodos.

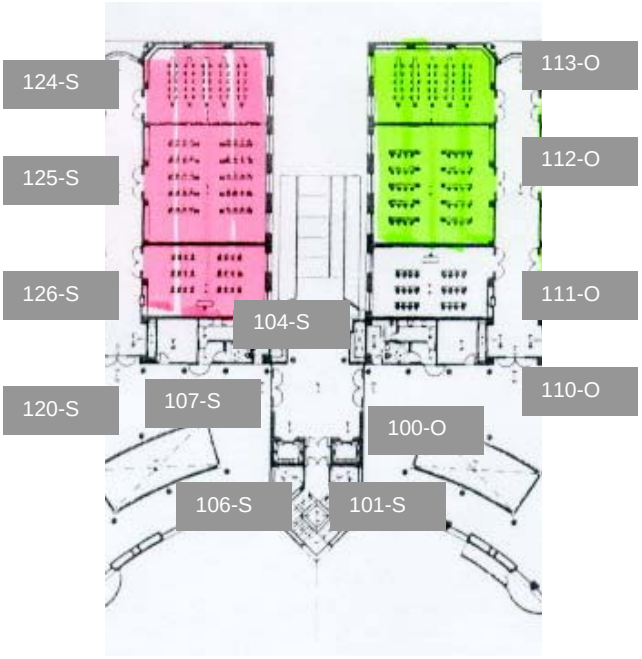
Los nodos del modelo de mapa implementado en el proyecto no son nodos virtuales, sino que tienen una representación física concreta: existen unas marcas colocadas en el edificio[10] en ciertas posiciones estratégicas de las

[10] Son las Marcas de Posicionamiento y Localización (MPL) diseñadas en [García-01]

salas del edificio (generalmente el en quicio superior de las puertas de acceso a la sala o al pasillo). Cada marca identifica al nodo asociado en el mapa topológico. Además, en estas marcas se basará todo el sistema de posicionamiento mediante visión, como se verá más adelante.

Finalmente, la implementación del concepto de mapa en este trabajo se corresponde con 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. Este fichero, del cual se da un ejemplo en la figura 3.5, contiene la sigiuente información en sus registros:

- Nombre del nodo correspondiente.
- Coordenadas X e Y absolutas de la marca (en centímetros escalados 1:500)
- Ángulo de entrada al nodo, que da una idea de la orientación que debe llevar el móvil al pasar por él.
- Distancia de aproximación al nodo, que ayuda a en las tareas de giro.



Ejemplo del fichero de Mapa

```
...
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
```

Figura 3.5. Ejemplo de asignación de nombres a nodos en el plano del mapa de la aplicación.

3.2. Visión

El segundo proyecto en el que se basa el presente trabajo de fin de carrera es el descrito en [López-02]. En él se implementaba un sistema de visión artificial que detectaba las marcas artificiales mencionadas antes en la imagen capturada por una cámara de video y, procesando esta imagen, obtenía la posición relativa de la cámara respecto de la marca. Toda la

implementación está basada, a su vez, en los trabajos realizados en la tesis doctoral [García-01].

A través de la API[11] del sistema videoforlinux2[12] basada en el controlador (o driver) original '*bttv*', diseñado por los hermanos Metzler, se consiguían capturar imágenes directamente desde una cámara de video en blanco y negro conectada a la tarjeta capturadora de video del sistema, permitiendo al usuario obtener la información del nivel de gris de cada píxel que compone la imagen. Para conseguir capturar imágenes con este controlador se debía disponer de una tarjeta con una determinada característica: que esté basada en el chipset Conexant bt848 o bt878. Tanto para el proyecto [López-02] como para el que detalla esta memoria se ha utilizado una tarjeta de la marca AverMedia que incorpora el chipset bt878.

La aplicación de captura de imágenes se basaba en una modificación del programa Xcaptest de forma que sea posible acceder al valor de los pixels que forman la imagen capturada. Así, los datos que se capturaban a través de la cámara eran mapeados en memoria y se mostraban por pantalla a la vez que eran procesados por el programa para obtener toda la información disponible de todas las marcas que se encontrasen en la imagen capturada. Esta información consistía en:

- ✓ Nombre del nodo, codificado en el código de barras de la marca.
- ✓ Posición relativa de la camara respecto al sistema de coordenadas de la marca (X,Y). Este sistema es el SCA en la figura 2.7.
- ✓ Ángulos de orientación de la cámara (α, β, γ) en el SCC (de nuevo, en la figura 2.7)
- ✓ Distancia entre la cámara y la marca.

3.2.1. Marcas artificiales.

Como ya se ha explicado, la estimación de la posición del móvil mediante visión artificial se basa en la detección y procesado de la información almacenada en unas marcas artificiales situadas en puntos estratégicos del entorno estructurado (nodos). Estas marcas fueron diseñadas por D. Juan Carlos García García en su tesis [García-01], y consisten en una hoja de tamaño DIN-A4 en la que se ha impreso lo siguiente:

- ✓ Cuatro círculos negros de un tamaño dado y situados en unas posiciones determinadas, próximas a las esquinas. La recuperación de los centroides de estos círculos será lo que permita obtener la posición y orientación relativa del móvil respecto de la marca. La técnica empleada

[11] Acrónimo inglés de *Application Programming Interface*, se refiere a la forma de uso de un conjunto de funciones de programación definidas en una biblioteca.

[12] Versión mejorada respecto de videoforlinux, el sistema de gestión de video del Sistema Operativo Linux.

pare la obtención de la posición se basa en recuperar la distancia relativa X e Y en el SCA, y a partir de estos datos obtener el ángulo de orientación del móvil respecto de la marca y su distancia en metros.

- ✓ La marca también incluye un patrón vertical de identificación, compuesto por un grupo de barras gruesas (grupo central-izquierdo), el cual es idéntico para todas las marcas, y permite su identificación en cada imagen. Este patrón está basado en un código Barker de 7 bits, cuya composición ha sido elegida para no ser confundido con otras estructuras formadas por el resto de objetos que se encuentren en la imagen.

- ✓ Por último, la MPL dispone de un código de barras de n dígitos, compuesto por un grupo de barras finas (grupo central-derecho), propio y distinto para cada marca, y que se lee en dirección vertical y en sentido descendente. Este código de barras será el que proporcione el valor de entrada a la base de datos de posición de todas las marcas dentro del edificio en cuestión (posición absoluta global en el SCG, figura 2.7).

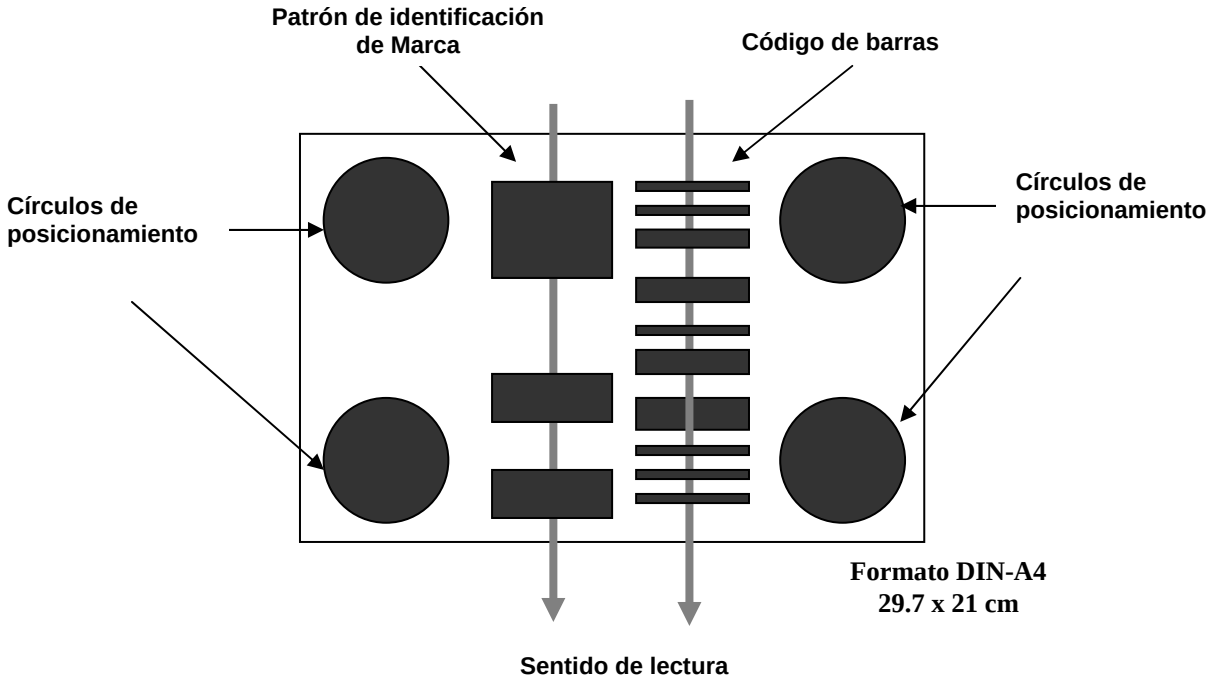


Figura 3.6. Estructura de una Marca Artificial.

3.2.2. Funcionamiento de la aplicación

La aplicación comenzaba tomando una de las imágenes capturadas por la cámara y la procesaba para detectar si existía en ella alguna marca artificial. Si es así, intentaba obtener la información codificada en las distintas marcas detectadas para calcular la posición absoluta del móvil en el instante de captura. Esto implicaba dos objetivos:

1. Calcular la posición relativa de la silla respecto de la marca.

2. Leer el código de barras para conocer el nombre del nodo al que correspondía la marca y así buscar en el mapa su posición absoluta.

Aunque se consiguiera el primer objetivo, esto no implicaba que el segundo fuera posible ya que la capacidad de decodificación del código de barras depende de la distancia a la que se encuentre la cámara de él. En cualquier caso, la aplicación no tomaba los datos de la marca para calcular la posición absoluta, sino que ofrecía el nombre del nodo, codificado en el código de barras de la marca, y la posición relativa de la cámara respecto del sistema de coordenadas de la marca. Con esta información, obtener la posición absoluta es casi inmediato, tal y como ya se expuso en el capítulo 2, aunque en esta aplicación no se llegaba a dar ese último paso.

El proceso completo puede estructurarse en las siguientes partes:

- ✓ Captura de la imagen.
- ✓ Reconocimiento de la/s marca/s artificial/es dentro de la imagen capturada mediante la búsqueda de los posibles códigos Barker.
- ✓ Extracción de los centroides de los cuatros círculos contenidos en las marcas artificiales detectadas.
- ✓ Lectura del código de barras y extracción del dato codificado.
- ✓ Obtención de los vectores de posición-orientación V1= (Xo, Yo, γ). Ver figura 2.7.

Para facilitar el proceso de la imagen, se introducía en una matriz de tamaño 640x480, de tal modo que cada columna de la imagen original se situase en una fila dentro de la matriz. Así, en la primera columna de la matriz estaría el comienzo de cada columna de la imagen: con esto se consiguía que la indexación de los datos se hiciera de una manera más rápida.

Con los datos ya preparados se llamaba al código de Detección de Marca, que recibía como parámetro de entrada la matriz con la imagen y que se dividía en las siguientes subtareas:

- ✓ Detectar la existencia de alguna marca artificial mediante la búsqueda de códigos Barker.
- ✓ En el caso de haber encontrado algún candidato a código Barker se pasaba juntarlos en grupos, en función de su tamaño y proximidad, y a su validación.
- ✓ Se calculaba la posición de los centroides de los círculos de cada marca detectada dentro de la imagen.
- ✓ Se procedía a la lectura de los códigos de barras de cada marca y a decodificar sus valores, en los casos en que fuera posible.

El siguiente paso del algoritmo principal, en caso de que se hubiese detectado alguna marca y se hubieran obtenido correctamente los centroides de los círculos, era realizar una llamada a la función de Conversión. Esta tarea recibía como parámetro de entrada las posiciónes de los centroides, medidas en número de píxels, y su cometido principal era convertirlas a coordenadas en metros.

Con estos datos se llamaba a una rutina que obtienia el vector posición-orientación M = (Xo, Yo, γ) de la marca respecto a la cámara, para cada marca.

Lo que sigue a continuación es una breve descripción del procesado de imagen, paso por paso.

Detección del código Barker.

Partiendo de la imagen capturada se debe encontrar alguna marca artificial. Su localización se basa en la detección de un patrón en la imagen que represente un código Barker.

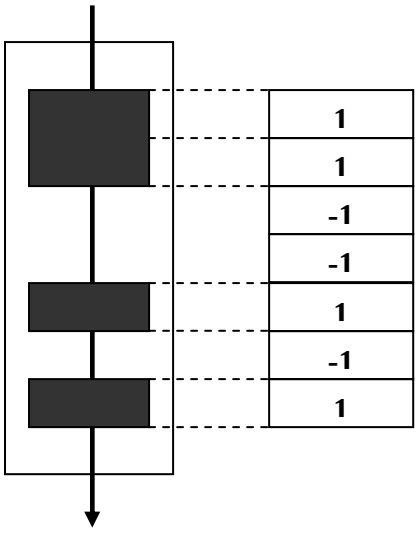


Figura 3.7: Estructura de la codificación de un código Barker.

El código Barker permite ser detectado con relativa facilidad ya que posee una forma que imposibilita su confusión con objetos que se puedan encontrar dentro de la imagen (persianas, baldas, etc), simplificando el procesado de la imagen y evitando la detección de falsos blancos. En nuestro caso se usa un código Barker de 7 bits con codificación directa, cuya estructura de código es [1,1,-1,-1,1,-1,1], siendo el 1 una barra negra y el –1 una blanca (ver figura 3.7). Este patrón presenta un valor de autocorrelación elevado que permite que sea identificado muy fácilmente si es leído en sentido descendente.

Ya que el código Barker es unidimensional, el algoritmo debe leer cada una de las columnas de las que consta cada imagen (640) y analizar si existe en toda ella algún posible candidato a ser código Barker.

Agrupación de códigos Barker en cluster.

Detectados los códigos Barker existentes en la imagen, el siguiente paso es agruparlos según su dimensión y proximidad y mediante el descarte de los grupos de códigos que no cumplan las siguientes condiciones:

- ✓ Para que se valide la detección de una marca artificial debe contener un grupo formado por dos o más códigos Barker. De esta manera se descarta cualquier espúreo en la misma.

- ✓ El código Barker detectado debe cumplir la relación de aspecto de sus barras, es decir, ancho entre alto. Así se consigue eliminar los ruidos debidos al código de barras de la marca.

- ✓ Finalmente, para evitar una detección falsa en el caso de que un código de barras cumpliera la relación de aspecto comentada, se eliminan todos los códigos detectados a la derecha de un grupo Barker mayor que estén dentro de la zona de cobertura de la marca.

Una vez validada la marca, el primer paso a seguir para obtener los centroides de los círculos de la marca será crear dos subventanas: una superior, que abarcará los dos círculos más altos, y otra inferior para los dos de abajo. El tamaño y posición de las dos subventanas se obtiene a partir de los parámetros de tamaño y posición del código Barker.

El paso siguiente es invertir el nivel de gris de los pixels contenidos en las subventanas. Los valores resultantes se normalizan entre 1 y 0 para después binarizar cada subventana con sólo dos niveles, asignando un 1 a aquellos valores que superen el umbral establecido (que en este caso es de 0.75) y 0 a los que no lo superen. Con este proceso se obtiene un mayor contraste entre el círculo y el fondo de la imagen.

Después de esto se crean unas miniventanas que contienen cada uno de los círculos. Para obtener estas miniventanas se detectan los flancos existentes entre los pixels que conforman el círculo y el fondo. Obtenidas las miniventanas de los círculos izquierdo y derecho dentro de las subventanas, sólo queda calcular los centroides de los círculos para tener los datos necesarios para obtener la posición del móvil respecto de la marca.

Cálculo de la posición relativa.

Una vez se han extraído todos los parámetros de información que contiene la marca artificial, la aplicación debe encontrar el vector de posición-orientación de la silla de ruedas a partir de las coordenadas de los círculos. Para ello es necesario realizar antes una conversión de las unidades de posición de los centroides obtenidos. En este proceso se parte del

conocimiento de los parámetros intrínsecos de la cámara, que han sido obtenidos a partir de un proceso de calibración off-line, y el objetivo es obtener los centroides calibrados en coordenadas en metros a partir de los centroides en medidas en número de píxels.

El paso final consiste en obtener el vector posición-orientación de la cámara (Xo, Yo, Zo, α, β, γ) a partir de las coordenadas de los centroides en la imagen (vease figura 2.7). Para realizar este cálculo se hacen las siguientes aproximaciones, teniendo en cuenta que la marca estará localizada siempre en un plano (X'Z') perpendicular al de rodadura (X'Y') tal y como se indica en [García-01]:

1. β nulo debido a que el plano de proyección de la imagen (X'Z') es perpendicular al plano de rodadura (X'Y').

2. α conocido pues ha de existir un pant-tilt controlado a bordo del robot que permita orientar el eje de proyección de la cámara. Las pruebas incluidas en la tesis, así como las realizadas en [López-02] y en el presente trabajo, se han realizado con α=90º, es decir, eje de proyección (z') paralelo al plano de rodadura (X'Y').

3. El problema termina siendo únicamente la obtención de una relación en 2 dimensiones, ya que no se tiene en cuenta la coordenada z del sistema de coordenadas global SCG, aunque ésta se podría obtener perfectamente con los algoritmos mencionados, expuestos en [García-01].

Para una descripción exacta de los algoritmos de posicionamiento y su implementación en este proyecto se puede consultar [García-01] y [López-02], respectivamente.

3.3. Simulación del EKF en Matlab.

El tercer y último trabajo en el que se basa este proyecto es el desarrollado en la trabajo de investigación [Marrón-02], en el que se realiza una simulación de un sistema con las especificaciones de este proyecto con Matlab y su módulo para trabajar con sistemas de Tiempo Real: *Real Time Workshop*. Así, se implementa el algoritmo del EKF en lenguaje C para Matlab y se simulan los resultados de su funcionamiento para la fusión de la información de odometría y visión en la navegación a lo largo de un trayecto.

3.3.1. Aproximaciones asumidas.

Para la consecución de los objetivos propuestos en este trabajo previo se establecieron unas condiciones de ejecución que se van a comentar a continuación ya que la mayoría de ellos serán utilizados en el presente proyecto.

Por un lado es necesario comentar el hecho de que los dos sistemas sensoriales no se desarrollan con el mismo periodo de repetición. Para tener en

cuenta esta diferencia se va a nombrar de distinto modo al periodo de muestreo (o de ejecución) de los dos sistemas. De este modo:

- ✓ T_{SO} , será el periodo de muestreo del sistema odométrico.
- ✓ T_{SV} , será el periodo de muestreo del sistema de visión, tal que $T_{SV}=n \cdot T_{SO}$ (son múltiplos).

El periodo de ejecución del estimador de posición será el del sistema más rápido, es decir, coincidirá con el de muestreo del sistema odométrico. La diferencia de los periodos de muestreo se verá reflejada solamente en la etapa de corrección del EKF, pues en la de predicción sólo entra en juego el modelo odométrico. Éste evolucionará a la misma frecuencia que la de muestreo del sistema odométrico.

La etapa de corrección solamente se ejecutará cada 'n' veces (siendo 'n' el número de veces que T_{SV} es múltiplo de T_{SO}), incorporando nuevas medidas del sistema de visión. Así los tres elementos (variables y matrices) que caracterizan al sistema de visión permanecerán constantes durante los $n \cdot T_{SO}$ segundos que dure su periodo de ejecución. En otras palabras, el algoritmo de fusión dará una salida solamente cada T_{SV}, aunque hará evolucionar el vector de estados odométrico de la etapa de predicción (que seguirá ejecutándose en todos los periodos de ejecución del algoritmo), cada T_{SO}.

3.3.2. Detalles de la implementación en Matlab

Las consecuencias de la diferencia de periodos de ejecución (T_{SO} y T_{Sv}) se observaron claramente en la simulación del algoritmo de fusión en [Marrón-02], y no fueron otras que la ralentización e incluso la divergencia en algunos casos del estimador de posición. Además, debido a este hecho, el ruido del modelo de odometría será especialmente importante, pues no se compensa mediante la etapa de corrección con tanta frecuencia y tendrá efecto durante más tiempo en el estimador. Este hecho puede llegar incluso a desestabilizar la convergencia de la ganancia de Kalman.

Otro aspecto importante que merece la pena destacar de la implementación que se hizo del EKF en el trabajo de investigación mencionado es la modificación del orden de la ejecución de las dos etapas del algoritmo de fusión. En cada periodo de ejecución se desarrollaba primero la etapa de corrección y luego la de predicción, que prepara el algoritmo para el siguiente periodo del estimador recursivo. Es decir, la implementación del EKF se hace en el orden inverso a como se ha desarrollado en este documento. En cambio, en el presente trabajo de fin de carrera se ha seguido el orden 'natural' del algoritmo.

4. Integración de Trabajos Anteriores

Durante este capítulo se va a explicar cómo se han integrado los desarrollos que sirven de base a éste Trabajo de Fin de Carrera en una única aplicación sobre sistema operativo Linux. En esta aplicación se pretende fusionar información de dos subsistemas sensoriales, odometría y visión, mediante un EKF. Este método permite estimar la posición del robot de forma robusta y, con ello, mejorar el proceso de navegación de la Silla de Ruedas Autónoma en el entorno estructurado de interés: el edificio de la Escuela Politécnica de la Universidad de Alcalá.

4.1 Implementación del Sistema en Módulos

La aplicación implementada en el proyecto [Marrón-00] en el que se basa fundamentalmente este Trabajo de Fin de Carrera siguía un esquema secuencial de ejecución de las tareas que la componían y que, a grandes rasgos, se podría describir como:

```
planificación del movimiento
bucle
{
    envío de consigna de movimiento
    espera de 50 mseg.
    comprobación del movimiento (odometria)
    calculo nueva consigna de movimiento
} hasta (fin del movimiento)
```

La inclusión de un algoritmo de fusión como el EKF no altera este esquema secuencial de ejecución ya que basta con ejecutarlo antes del cálculo de la nueva consigna de movimiento (o, incluso, considerarlo como parte de éste). El problema aparece en el hecho de que la fusión implica una nueva

tarea que aporte datos de visión que sean fusionados con los obtenidos mediante dead reckoning. Este hecho es el que imposibilita el esquema de ejecución que se venía utilizando en la aplicación.

La ejecución de la aplicación de posicionamiento mediante visión artificial, tal y como se comenta en la memoria del desarrollo original [López-02], es muy intensiva en cálculo y, además, de duración muy superior a un periodo de odometría y, por tanto, de ejecución del EKF. Como ya se observó en el comentario sobre la simulación de este proyecto en el trabajo previo [Marrón-02] (ver capítulo anterior), el funcionamiento de la aplicación ejecutará sólo la etapa de predicción del EKF utilizando para ello datos obtenidos mediante odometría con un periodo de repetición de 50 milisegundos, y sólo se aplicará el algoritmo completo (predicción y corrección) cuando se disponga de datos de posición proporcionados por el procesado de la información de visión artificial.

Por tanto, es necesario encajar la ejecución de la tarea de visión en este esquema. Parece obvio que la mejor forma de aprovechar el desarrollo existente es utilizar el tiempo muerto en el que la aplicación no hace nada sino esperar al comienzo del nuevo ciclo de control (periodo 50 ms) mientras la silla se mueve. Sin embargo, esta forma de resolver el problema, requiere replantear el modelo de ejecución de la aplicación completa.

Para comprender la problemática abordada se debe comentar que la política de planificación por defecto que un sistema operativo de Tiempo Compartido asigna a los procesos que ejecuta intenta que todos ellos dispongan del mismo tiempo de ejecución en el procesador. Para lograrlo, se divide el tiempo global de ejecución en bloques y éstos se van asignando a cada proceso de forma cíclica. Desafortunadamente, este modelo de planificación no es válido para conseguir el objetivo planteado ya que se necesita que las tareas se ejecuten en un orden que establecido de antemano y con unas temporizaciónes más o menos estrictas, no permitiendo que sea el sistema operativo el que decida cuándo y cuánto tiempo se está ejecutando cada tarea. Estas consideraciones no eran aplicables al antiguo esquema de la aplicación ya que sólo se ejecutaba un proceso y se podía suponer que el Sistema Operativo no ejecutaba ninguno más a la vez. Por tanto, no intentaba repartir el tiempo de proceso con ninguna aplicación más o, si lo hacía, su influencia era mínima.

La solución adoptada consiste en separar la aplicación en módulos ejecutables con políticas de planificación y prioridades de ejecución distintas. De esta forma se consigue la ejecución en el orden y con la duración deseada. Estos módulos serán de ejecución casi independiente y se comunicarán entre sí empleando los mecanismos que para ello provee el Sistema Operativo: señales y colas de mensajes.

La división en módulos, representada en la figura 4.1, se hace de la siguiente forma: se divide la aplicación principal en un Módulo de control Principal que se encargará de gestionar todas las tareas y que incluirá el código del EKF. Para implementarlo se partirá del código de la aplicación de

Navegación desarrollado en [Marrón-00] y se añadirá la nueva funcionalidad (EKF, gestión de procesos rediseñada y comunicaciones).

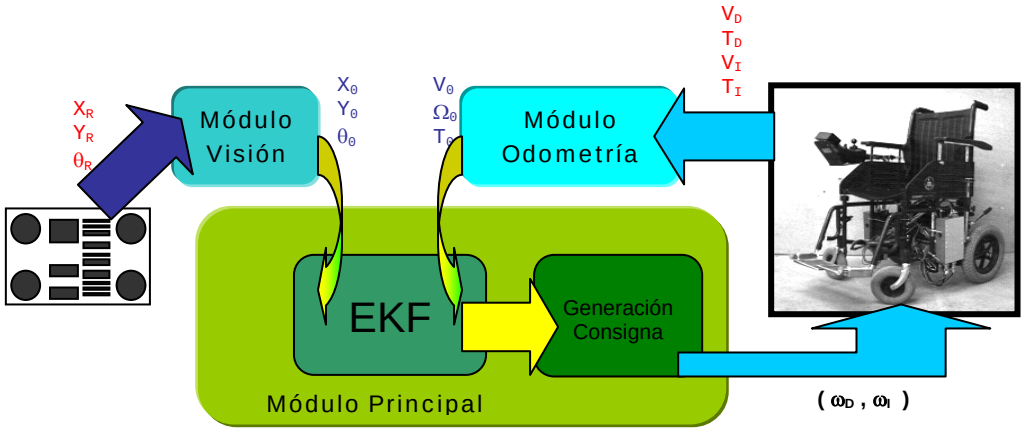


Figura 4.1. Esquema de la aplicación y paso de mensajes

De este Módulo Principal se escindirá el Proceso de Odometría, que pasará a formar parte de un módulo independiente que se comunicará con el principal. La aplicación de Visión, por su parte, también formará otro módulo secundario.

El hecho de separar el proceso de odometría pretende conseguir una coherencia en el desarrollo de la aplicación: los subsistemas que proporcionan medidas de posición del robot son módulos independientes que se comunican con el principal. De esta forma se consigue que en un futuro sea más sencillo sustituir un módulo de medidas por otro distinto (o mejorado) o, incluso, añadir alguno más.

Con todo ello, las tareas de las que se encargará el módulo principal se resumen como sigue:

1. Iniciar a los demás módulos.

2. Diseñar la ruta de navegación entre el nodo origen y destino.

3. Enviar consignas al control de bajo nivel de la silla para que siga el camino marcado.

4. Gestionar la ejecución de los procesos de odometría y visión y recoger sus resultados.

5. Fusionar la información recibida para estimar la posición actual de la silla y en base a ella, continuar el movimiento.

6. Al alcanzar el destino, parar el movimiento y liberar todos los recursos indicando también a los otros dos procesos que hagan lo mismo.

En cuanto a los módulos de odometría y visión, su misión consistirá únicamente en inicializarse, obtener información de la posición y enviarsela al módulo principal. Finalmente, cuando éste lo indique, liberaran los recursos ocupados y finalizarán su ejecución.

Con esta división, el módulo de control principal será el que se ejecute con una prioridad mayor y lo hará cada 50 ms. Este proceso será el que se encargue de asegurar la ejecución del módulo de Odometría cada 50 ms. y contiene al EKF. La prioridad inmediatamente inferior corresponderá al módulo de Odometria, para forzar al planificador del sistema operativo a ejecutar este proceso siempre que el Principal no tenga que ejecutarse. El proceso de Odometría tiene el mismo periodo de ejecución que el EKF y una duración muy pequeña en comparación con su periodo. Por último, la tarea menos prioritaria será la de Visión, que aprovechará el tiempo en el que no se ejecuta ninguna de las otras dos aplicaciones para ir realizando su trabajo, como ya se ha explicado. Cuando acabe, avisará al módulo de control Principal enviando su información de posición.

Tanto la planificación como los detalles de implementación serán ampliados en el capítulo siguiente, dedicado a la descripción del funcionamiento real del sistema.

4.2 Consideraciones de Tiempo Real.

En el trabajo previo [Marrón-00] se discutía la necesidad de que la aplicación se ejecutara bajo especificaciones de tiempo real estricto. Al final, la conclusión que se obtenía en ese trabajo era que, teniendo en cuenta que las exigencias de la aplicación no eran muy altas a nivel de tiempo de cómputo y que no existían tareas que requiriesen una ejecución de forma concurrente, no era preciso un ajuste estricto a los requisitos de tiempo real.

Sin embargo, debido a la ampliación de funcionalidad de la aplicación acometida en este proyecto y su nuevo funcionamiento modular se produce un cambio de condiciones que exige replantearse esta cuestión.

El hecho de que en la nueva aplicación, además de tener un periodo de ejecución que cumplir (los 50 ms. del algoritmo de control y del EKF), haya que compartir el procesador con el subsistema de Visión (que se ejecutará independientemente cuando se lo permitan los otros dos módulos) hace temer por el cumplimiento de el periodo de ejecución comentado, ya que debe garantizarse una gestión de procesos eficaz, una temporización precisa y un tiempo de cambio de tareas pequeño y, además, acotado.

Se puede constatar en la amplia literatura existente sobre ello que Linux no es un sistema operativo que pueda asegurar un funcionamiento en tiempo real estricto. Esto es debido a que el código de su núcleo no es interrumpible, en otras palabras, si se está ejecutando el núcleo del Sistema Operativo las aplicaciones de usuario deben esperar inevitablemente a que termine y su tiempo de ejecución no está acotado: no podemos hablar de tiempo real. Sin embargo, sí podemos presumir que, si nuestra aplicación va a ser la única que

se esté ejecutando en ese momento en el sistema, el tiempo de ejecución de rutinas del kernel[13] no va a afectarnos demasiado y, si bien no podemos asegurar cuanto, si podemos decir que la temporización no se desviará mucho de las espectativas planteadas.

Por tanto, la aproximación realizada al tiempo real será válida sólo en condiciones de carga del sistema nula o muy baja, de tal forma que podamos considerar que sólo se ejecuta la aplicación diseñada. Es importante señalar que, para futuras ampliaciones de este proyecto, sí será necesario utilizar un núcleo de sistema operativo de Tiempo Real que garantice las condiciones necesarias para que el periodo de ejecución de 50 ms. se siga cumpliendo.

Para finalizar, se describirá ahora los métodos proporcionados por Linux para hacer la aproximación al funcionamiento en Tiempo Real lo más efectiva posible. Ya que no se pueden garantizar todas las condiciones necesarias para la ejecución en Tiempo Real, se intenta una aproximación lo más cercana posible haciendo uso de los siguientes recursos:

- ✓ Bloqueo de páginas en memoria: Se fuerza al sistema operativo a que las páginas de memoria asignadas a los procesos de la aplicación sean persistentes y no pasen a almacenamiento secundaro (disco duro). De esta forma se evita el mecanismo de memoria virtual. La memoria virtual introduce no determinismo en la aplicación ya que el tiempo que se tarda en resolver un fallo de página no está acotado.

- ✓ Planificación RT[14] con prioridades de ejecución: como se ha comentado, este mecanismo no es infalible debido a la no interruptibilidad del kernel pero sí permite asegurar la planificación entre los procesos de usuario.

- ✓ Temporización mediante relojes de RT: De esta forma se obtendría una precisión de nanosegundos. Sin embargo, debido a que la referencia temporal utilizada no es otra que el reloj del ordenador y la precisión obtenida alcanza los 100 microsegundos.

- ✓ Comunicación mediante señales RT: Utilizando señales de Tiempo Real aseguramos que éstas no van a perderse, sino que serán todas encoladas y, además, aceptadas en orden de prioridad.

4.3. Módulo principal de Control del Sistema

El primer paso para comenzar a adaptar los trabajos previos a la aplicación objetivo de este Trabajo Fin de Carrera debe ser, sin duda, la aplicación de navegación que pasará a vertebrar el módulo principal del sistema.

[13] El núcleo del sistema operativo se conoce también como kernel, que es la traducción de esta palabra al alemán. En este documento se utilizarán ambas.

[14] Acrónimo de Real Time, es una forma de referirse al Tiempo Real

El código implementado en [Marrón-00] se ha modificado para que se ejecute sobre una plataforma Linux y se ajuste al aumento de funcionalidad acometido en este proyecto. La transformación se estructuró en 2 líneas fundamentales: física y lógica.

- ✓ Física: La gestión de dispositivos hardware es totalmente distinta entre la plataforma origen (Windows) y la de destino (Linux). Es necesario ajustar el código que se encarga de manejar la plataforma física (comunicación con el control de bajo nivel de la silla de ruedas) al nuevo sistema operativo.

- ✓ Lógica: Como ya se ha explicado, el aumento de funcionalidad requiere una replanificación de cómo y cuándo se ejecuta cada proceso, de qué tiempo dispone cada uno, cuál es su importancia relativa, etc. Esto requerirá prácticamente la re-escritura de gran parte del código de la aplicación para ajustarse a unos requisitos de Tiempo Real más estrictos e implementar mecanismos de comunicación y gestión de procesos.

  Mientras que la transformación lógica no se explicará aquí porque queda suficientemente explicada en el capítulo siguiente, sí se va a detallar los cambios a nivel físico.

4.3.1 Nivel fisico

Tal y como estaba codificada la aplicacion original de control se utilizaba un controlador que, desde el espacio de usuario[15], se comunicaba con el Neuron Chip encargado de gestionar el control de bajo nivel de la silla escribiendo en una memoria Dual Port que actuaba de interfaz entre ambas plataformas. Los datos de odometría enviados por el Neuron también se escribían en esta memoria para que la aplicación de navegación los leyese. Este controlador era una adaptación del que diseñó Johann Persson en [Persson-99].

En la primera aproximación al problema se intentó modificar el código existente para que funcionara sobre Linux. Esto no funcionó debido a que, como se ha dicho antes, la gestión, configuración y uso, del hardware es muy distinta entre Linux y Windows. En este caso, la forma de configurar el puerto paralelo para realizar la comunicación en modo EPP con la tarjeta difería notablemente.

Se realizó una re-escritura del controlador original basada en el código de un controlador para gestionar scaners conectados al PC con Linux a través del puerto paralelo. Este controlador, junto con su documentación, permitió que obtuviese una aproximación muy fiel al controlador original, utilizando el modo

[15] En teoría de Sistemas Operativos se distingue entre nivel de ejecución en espacio de usuario, en el que la ejecución de determinadas acciones está limitada o, al menos, controlada; y espacio del núcleo, en el que se puede realizar cualquier acción sobre el ordenador sin restricciones.

EPP del puerto paralelo mediante escrituras directas al puerto. Sin embargo, este controlador no cubrió totalmente la funcionalidad esperada.

El controlador implementado accedía a la memoria Dual Port: escribía y leía en ella en las posiciones que se le indicaba. Sin embargo, en algunas posiciones no lo hacía correctamente ya que presentaba errores en algún bit, y no era capaz de acceder al sistema de semáforos de la Dual Port. Realizando una comprobación exhaustiva del movimiento de las líneas durante las transmisiones se descubrió que algunas de las señales estaban muy contaminadas por ruido y que, por eso, la comunicación entre el PC y la memoria no era satisfactoria. La presencia de ruido se puede achacar a que el driver se diseñó sobre un PC más antiguo, con un hardware de control del puerto paralelo distinto. Los puertos actuales soportan mayores velocidades de transmisión a costa de un filtrado menos estricto de la señal y éste, para la aplicación y estado del hardware, es insuficiente.

La resolución de este problema requería un rediseño del hardware de comunicaciones. Se optó por eliminar la tarjeta de interfaz mediante Dual Port del esquema y realizar la comunicación directamente con el Neuron Chip. Esta estrategia ya había sido utilizada satisfactoriamente por Jesús Nuevo para el robot Alcarroby en [Nuevo-04]. De esta forma, se debía cambiar el conexionado con la placa que contiene al Neuron. En el Anexo A se puede ver un esquema con la correspondencia de líneas entre el puerto paralelo y el Neuron.

La implementación software del driver propuesto por Jesús Nuevo consistía en un módulo cargable en el kernel de Linux en lugar de un programa controlador ejecutándose en espacio de usuario. Así, el acceso al Neuron se realizaba mediante la lectura y escritura (utilizando la interfaz de llamadas al sistema) de un fichero especial de dispositivo[16] que representaba al Neuron Chip y era creado durante la instalación del driver.

El driver utilizado en Alcarroby se ajustaba a la perfección a las nuevas necesidades excepto por la forma de implementar la operación de lectura del Neuron Chip. Existe una peculiaridad en el funcionamiente de éste, y consiste en que si está enviando datos, en este caso hacia el PC, y el PC no los lee en un determinado tiempo (0.84 segundos) el chip se resetea y requiere una resincronización de la comunicación. Para evitar esta situación, la solución planteada en [Nuevo-04] es que el driver no lea del Neuron Chip en el momento en que se ejecute una llamada a `read()`, como sucedería en un controlador de dispositivo típico. En su lugar, el driver sondea al dispositivo cada cierto tiempo y realiza una lectura cuando hay datos disponibles, almacenandolos en un buffer interno del driver. Será el contenido de este buffer el que recibe el usuario cuando realiza una lectura sobre el dispositivo.

[16] En Linux, la comunicación con el hardware se realiza como si estuvieramos escribiendo en un fichero. Este fichero representa al dispositivo con el que nos comunicamos, y la gestión de la comunicación es responsabilidad del sistema operativo, consiguiendo de esta forma que el proceso sea transparente al usuario.

Sin embargo, este modelo de funcionamiento no es válido para la aplicación aquí desarrollada porque requiere la ejecución de código del kernel en momentos puntuales y sin control por parte de la aplicación.

Teniendo en cuenta los requisitos estrictos de temporización de la aplicación desarrollada y que todo el tiempo de proceso es necesario para la aplicación, el modelo de lectura basado en sondeo y buffer no era viable. Por tanto, se recodificó la operación de lectura de tal forma que leyese del Neuron en el momento de la llamada. Teniendo en cuenta que en nuestra aplicación se hará una lectura cada 50 milisegundos, no habrá lugar para un *time-out* del Neuron Chip.

Se adaptaron y probaron dos versiones del driver: una que accedía al puerto paralelo usando la API del driver del puerto paralelo que incluye el kernel de Linux y otra que leía y escribía directamente en el puerto. Al final, la que funcionó de la forma esperada fue esta última, ya que la primera no realizaba bien la espera a los cambios de estado de las líneas de control del puerto, necesarios para el correcto seguimiento del protocolo de comunicación.

Por otro lado, al prescindir de la memoria Dual Port en la comunicación entre el Neuron y la aplicación se tuvieron que realizar algunos ajustes en el protocolo de comunicaciones de alto nivel. Como ya se ha comentado, el funcionamiento en la plataforma original consistía en que el PC escribía sus consignas de control en la memoria Dual Port y el Neuron las leía y hacía efectivas. A su vez, el Neuron escribía en la memoria los datos de velocidad y tiempos de medida y la aplicación los leía cuando le parecía oportuno, de forma asíncrona.

Esta forma de comunicación ya no es posible al no haber interfaz, así que se diseñó una nueva solución, consistente en el siguiente protocolo:

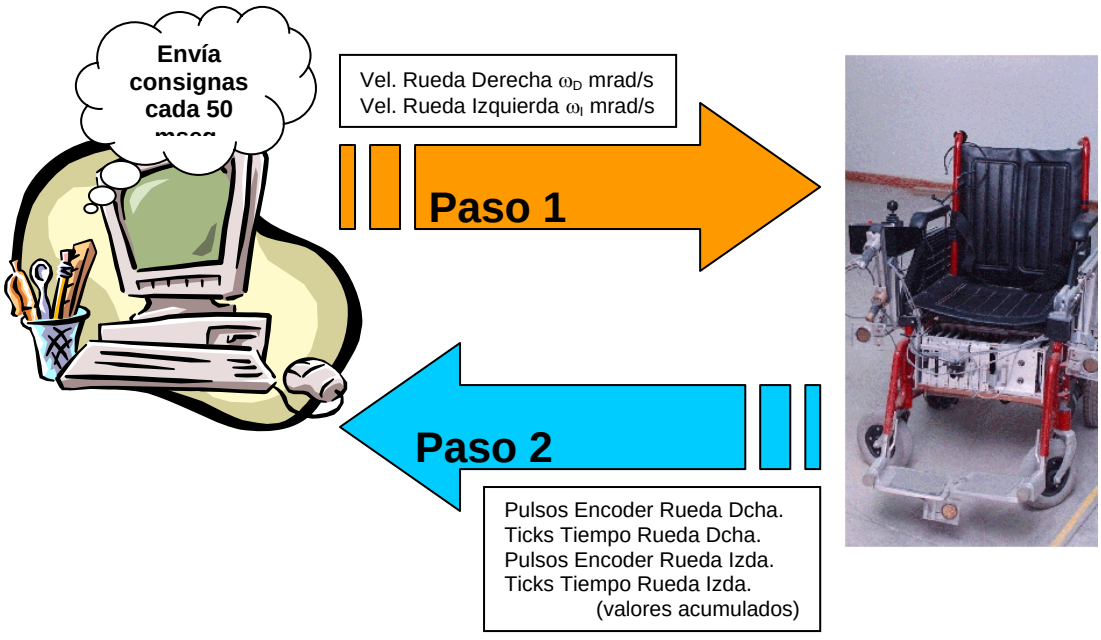


Figura 4.2. Esquema del protocolo PC – Neuron Chip de Alto Nivel

La aplicación envía las consignas de velocidad al Neuron, éste las lee, las hace efectivas y, además, la recepción de consignas indica al Neuron que responda con los datos obtenidos de velocidad y tiempo acumulados.

Así, finalmente, se consigue una comunicación eficaz y síncrona entre la aplicación y el PC.

4.4. Módulo de visión.

Para adaptar él código de visión a los propósitos descritos en este trabajo únicamente hubo que completar el cálculo de posición relativa de la cámara respecto de la marca con el de posición absoluta del móvil y eliminar algunas características innecesarias que aumentaban su tiempo de ejecución.

Uno de los cambios realizados consistió en limitar las espectativas de la aplicación en cuanto a búsqueda de marcas. Como se explicaba en el capítulo de Trabajos Previos, la aplicación de visión buscaba e intentaba obtener la información de todas las marcas que apareciesen en la imagen capturada. Esto no es necesario ya que para obtener la posición es suficiente con la información contenida en una de ellas. Se determinó que la marca elegida fuera aquella cuyo código Barker tuviera mayor tamaño en la imagen, para así facilitar también el proceso de decodificación del código de barras.

Si el código de barras no tuviera la longitud suficiente en la imagen para ser decodificado correctamente, a pesar de estar procesando el código de barras de mayor longitud en la imagen, la información que nos puede aportar la marca no nos sirve porque no es completa. Sin ese dato no se puede obtener la posición absoluta. Por lo tanto, en un caso como este se desecharía la imagen y se pasaría a procesar una nueva. En un trabajo futuro se podría implementar un mecanismo para que la aplicación deduzca qué marca está viendo sin tener que leer el código de barras, en base a la información que tiene de la ruta diseñada.

También se comentó en la descripción de la aplicación en la que se basa este módulo que los cálculos se detienen en la posición relativa de la cámara con respecto al sistema de coordenadas de la marca. Sin embargo, la información posicional que interesa para la aplicación es la absoluta, por lo que el cálculo ha de ser completado. Para ello se necesita la información sobre la posición absoluta de la marca contenida en el mapa del entorno.

La posición absoluta de la marca queda descrita por la terna (X , Y , θ) en el SCG, donde X e Y son las coordenadas posicionales de la marca en el mapa del edificio y θ es el ángulo que forma la perpendicular a la marca con el eje X del SCG. Esto está representado en la figura 4.3.

El problema viene dado porque en el mapa hay una marca pero, físicamente, podemos verla como dos: la que se ve al entrar en un nodo y la que se ve al salir. Se trata de la misma marca vista por delante y por detrás pero, ¿cómo saber qué angulo usar? ¿ θ ó (θ -180º) ? En el mapa del entorno,

el ángulo θ es el que forma el eje ‘X’ del SCG con la sección de perpendicular visible desde un nodo de jerarquía mayor. Así, por ejemplo, en la marca que aparece en el acceso de una sala a un pasillo el dato de θ que aparece en el mapa se referirá a la orientación que debe tener la silla, ángulo que forma su vector de velocidad lineal con eje X del SCG, para acceder de la sala al pasillo.

Para simplificar el cálculo de jerarquía de nodos que lleva al discernimiento del ángulo adecuado para los cálculos se han hecho las siguientes suposiciones sobre el nodo de partida:

- ✓ Si el nodo inicial es una sala (ABC, con C≠0. P.E.: 349), la silla se encuentra dentro de la sala
- ✓ Si el nodo inicial es un pasillo (ABC, con C=0. P.E.: 340), la silla se encuentra fuera del pasillo.
- ✓ Si el nodo inicial es una entrada a planta (A00. P.E.: 300), la silla se encuentra fuera de la entrada, mirando hacia el interior.

Con estas suposiciones previas respecto al origen del movimiento, es sencillo implementar una máquina de estados para descubrir qué ángulo tomar en función de la jerarquía del último nodo visto y del actual. Otra solución, que se deja aquí como propuesta, es utilizar el mismo código Barker actualmente utilizado en la marca para aquellas marcas que se encuentren en el interior de una sala y otro código para las situadas en el exterior. Si este código fuera, por ejemplo, el mismo que se utiliza girado 180º en el plano del papel la implementación de su detección en la aplicación actual sería muy sencilla.

Hechas todas estas consideraciones, las ecuaciones que rigen el paso entre coordenadas relativas y absolutas son las siguientes:

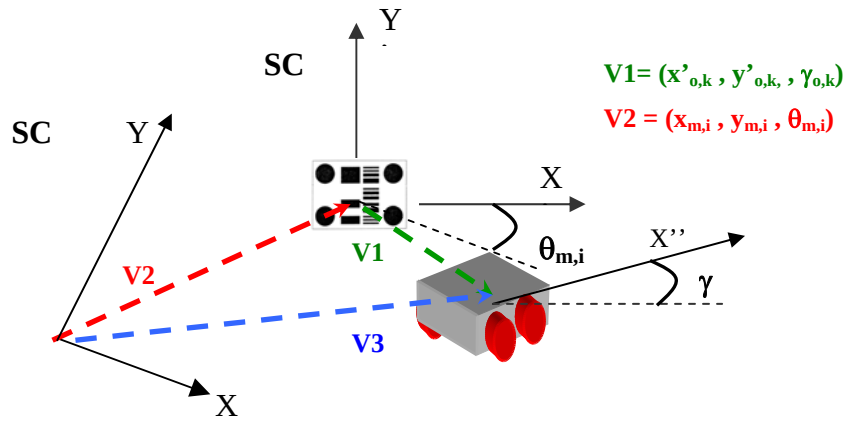


Figura 4.3. Representación 2D de la transformación a aplicar a la salida del algoritmo SPL ($\vec{z}'_{v,k}$)obtener la posición absoluta del móvil (salida del algoritmo global de visión $\vec{z}_{v,k}$)

$$\vec{V}3 = \vec{V}1 + \vec{V}2$$

$$\begin{aligned} x_{v,k} &= x_{m,i} + x'_{0,k} \cdot \cos(\theta_{m,i}) - y'_{0,k} \cdot \sin(\theta_{m,i}) = x_{m,i} + x'_{v,k} \cdot \cos(\theta_{m,i}) - y'_{v,k} \cdot \sin(\theta_{m,i}) \\ y_{v,k} &= y_{m,i} + x'_{0,k} \cdot \sin(\theta_{m,i}) + y'_{0,k} \cdot \cos(\theta_{m,i}) = y_{m,i} + x'_{v,k} \cdot \sin(\theta_{m,i}) + y'_{v,k} \cdot \cos(\theta_{m,i}) \\ \theta_{v,k} &= \theta_{m,i} + \theta'_{v,k} = \theta_{m,i} + \gamma_k \end{aligned} \quad <4.28>$$

4.5. EKF

Se ha dicho más arriba que la implementación del EKF se ha incluido en el módulo principal de Control y Navegación. Efectivamente, se ha planteado su diseño como funciones dentro de este módulo ya que su implementación como módulo aparte no aportaba ninguna ventaja.

A diferencia de su primera implementación sobre la plataforma Matlab, como una única función que implementaba todo el algoritmo y que, además, invertía el orden de sus fases como ya se comentó en el capítulo 3, el modelo de implementación elegido en este trabajo de fin de carrera es el de una función envoltorio que se encargará de ejecutar, según corresponda, sólo la fase de predicción o la predicción y luego la corrección, habiendose implementado cada fase en una función aparte. Con este enfoque se consigue un código y un funcionamiento más estructurado, lo cuál facilita su revisión y mejoras.

El gran problema a resolver al portar código producido para Matlab a otra plataforma es que, la mayoría de las veces, la plataforma destino carece de la potencia y de todas las funciones de Matlab para manejo de matrices.

Esto queda parcialmente solucionado con el diseño que se realizó en la implementación de [López-02] para cálculo mátricial. En ese trabajo se definió un tipo de datos matricial (`tMatriz`) y se implementaron la gran mayoría de funciones necesarias. Lo que se ha hecho en este trabajo es sacar ese código de la aplicación de visión para pasar a implementar con él una librería común para trabajo con matrices basada en el tipo de datos `tMatriz`. Así, ampliando las funciones ya programadas con alguna más, este desarrollo es compartido por el código de visión y el del EKF. Se puede encontrar una descripción detallada de las funciones de esta librería y del tipo de datos `tMatriz` en el capítulo 6.

5. Descripción del Funcionamiento del Sistema

A lo largo de este capítulo se va a describir cómo transcurre el funcionamiento normal de la aplicación diseñada. En primer lugar, se describe la sencilla interfaz de usuario diseñada, para luego dar paso a explicar como se desarrolla la inicialización del Sistema. Después de eso se pasa a exponer cómo se han resuelto finalmente las cuestiones relativas a la planificación de procesos descritas en el capítulo anterior y cómo se realiza la comunicación entre módulos. Tras estas explicaciones, el capítulo finaliza con la descripción del funcionamiento de la aplicación durante el movimiento de la silla a lo largo del trayecto.

5.1.Interfaz de usuario

La interfaz diseñada para la interacción del usuario con el sistema es muy sencilla, ya que no es objetivo de este trabajo ni su diseño ni optimización. Consiste en una comunicación, mediante la linea de comandos, del destino que se pretende alcanzar y del fichero de mapa que ha de ser utilizado.

Por lo tanto, la aplicación comenzará a ejecutarse cuando introduzcamos en la shell del sistema operativo una orden como esta ,previamente situados en el directorio donde se encuentren los ejecutables de la aplicación y accediendo como superusuario (esto es, como usuario 'root'):

navega nodo_destino fichero_mapa

Por ejemplo:

```
root@PC-silla# navega 102 EpOeste.map
```

El hecho de que la aplicación deba ejecutarse con privilegios de superusuario se debe a que éstos son necesarios para poder realizar algunas de las tareas que dan cumplimiento a la 'aproximación al tiempo real' de la que ya se ha hablado en esta memoria: gestión de póliticas de planificación, asignación de prioridades a procesos y bloqueo/desbloqueo de páginas de memoria de un proceso sólo pueden ejecutarse si el usuario que lanza el proceso es 'root'.

Una mejora que se ha realizado de la interfaz que presentaba la aplicación de navegación original diseñada en [Marrón-00] es que ya no es necesario indicar el nodo de partida en la llamada a la aplicación, sino que este dato lo obtiene por sí solo el sistema buscando la información en alguna marca artificial del entorno mediante visión artificial. Sin embargo, sí sigue siendo necesario indicar el fichero que contiene el mapa de nodos del edificio en el que se encuentra la silla y, con esto, damos algo de información previa sobre el origen del movimiento.

Como la situación deseable sería que el usuario no tenga que conocer dónde se encuentra, una posible mejora a abarcar en futuros proyectos sería codificar, de alguna forma, la información referente al edificio en el que nos encontramos en las marcas artificiales. Así, sería la aplicación la que facilmente podría decidir qué fichero de mapa utilizar.

Este podría ser un paso intermedio para solucionar el problema ya que, en futuras ampliaciones del proyecto, lo que se pretende es que el mapa del edificio en cuestión se descargue en el movil de alguna forma transparente al usuario. Por ejemplo, se podría transmitir vía radio al entrar en el edificio en cuestión.

5.2 Inicialización

En su fase de inicialización, lo primero que hace el programa principal es abrir el fichero del mapa que se pasó como argumento en la llamada al ejecutable. Hecho esto, se abre la cola de mensajes por la que se realizará la comunicación con los otros procesos y se indica qué funciones manejarán las señales que se reciban durante la ejecución.

Con estas inicializaciones previas, se pasa a bloquear las páginas de memoria asignadas al proceso (actuales y futuras) para que no pasen a almacenamiento secundario y se combia la política de planificación del proceso a FIFO, esto es, mientras no haya algún proceso de prioridad mayor, cuando éste tome el procesador no dejará de ejecutarse hasta que haya finalizado o quede en espera de algún evento. Además, asignamos a este proceso principal la máxima prioridad posible para procesos de usuario.

Esta sería la inicialización básica que, con diferencias mínimas, se ejecutará también al iniciar los procesos de odometría y visión:

- ✓ Apertura de cola de mensajes.
- ✓ Gestión de señales a utilizar.

- ✓ Bloqueo de páginas en memoria principal.

- ✓ Establecimiento de política de planificación y prioridad del proceso.

Continuando con el proceso principal, el siguiente paso será llamar al gestor de trayecto, que será quien se encargue del arranque y funcionamiento de toda la aplicación. Lo primero que hace este gestor es iniciar el proceso que se encargará de la visión artificial, comunicándole en el arranque qué fichero de mapa debe utilizar para obtener la información de posición. Se arranca primero este proceso debido a que será él quien descubra cuál es la posición inicial del móvil, permitiendo así que continúe el funcionamiento normal de la aplicación. Si no se pudiera conocer el lugar de origen del que partimos no se podría continuar.

Llegados a este punto, aunque se ha arrancado el proceso de visión, éste no ha comenzado todavía a ejecutarse. Esto se debe a que se ha forzado una planificación FIFO para el proceso principal, y éste no cederá el procesador hasta que finalice o quede esperando algún evento de forma inactiva.

Por lo tanto, la solución tomada para que otro proceso pase a la acción en un momento dado será dormir el proceso principal un tiempo superior al necesario por el otro proceso para cumplir su misión y, después, devolverle el procesador despertándole mediante el envío de una señal (recuerdese que, al inicio, se especificaron las funciones que debían ejecutarse a la llegada de determinadas señales).

El tiempo total que el proceso principal está dormido nunca debe agotarse completamente, debe ser interrumpido mediante el mecanismo de señales. Esto se asegura tomando un tiempo alto en relación con los tiempos de ejecución de la aplicación y, para ésta, se consideró que 5 segundos es tiempo más que suficiente.

Este será el mecanismo utilizado a lo largo de la ejecución de la aplicación para realizar gestionar el orden y tiempo de ejecución de los procesos. Sin embargo, para esta primera vez, lo que hace el proceso principal es esperar que el proceso de visión le devuelva, mediante un mensaje, el resultado de su primera iteración con la posición inicial del movil. Por lo tanto, el proceso queda bloqueado en esta operación de entrada/salida y así el proceso de visión tiene vía libre para ocupar el procesador.

5.2.1. Inicialización del proceso de visión.

Como se dijo antes, la inicialización básica de los tres procesos que implica la aplicación (principal, visión y odometría) es muy parecida. Así, las primeras tareas que realizará este proceso serán:

- ✓ Abrir la cola de mensajes a través de la cual se realizará la comunicación con otros procesos.

- ✓ Indicar al sistema operativo cuáles serán las funciones que gestionarán determinadas señales que recibirá el proceso a lo largo de su ejecución.

- ✓ Bloquear las páginas de memoria asignadas al proceso para que no pasen a almacenamiento secundario.

- ✓ Asegurar que la política de planificación de este proceso será la que el sistema operativo asigna por defecto a cualquier proceso, sin prioridad de ejecución sobre ningún otro y con tiempo de proceso limitado, compartiendo el procesador. La justificación y explicación de todas las planificaciones se hará más adelante.

- ✓ Abrir el fichero que contiene el mapa del edificio.

Después de esta inicialización general quedan las operaciones propias del proceso para que esté listo para el funcionamiento. Estas consisten en la apertura del dispositivo de captura de imágenes, esto es, la tarjeta capturadora de señal de TV y su posterior configuración para que tome como entrada de señal la correspondiente a la cámara de video ya que, por defecto, el driver de la tarjeta toma la señal de televisión. La última operación importante de configuración consiste en indicar al driver las características de imagen que necesitamos: altura y anchura de la imagen, formato de pixel, etc.

Con todo esto, el proceso de visión está listo para realizar su función y entra en un bucle de ejecución infinito que sólo se romperá cuando reciba la señal adecuada por parte del proceso principal. La ultima operación de este bucle consiste en dormir al proceso, en espera de que el proceso principal le indique, mediante otra señal, que debe realizar otra iteración completa de visión. Una iteración normal consiste en la captura de una imagen mediante la cámara y en su proceso para obtener como resultado las coordenadas de posición absoluta de la silla, que se devolverán mediante un mensaje al proceso principal, despertándole para que continúe así con la inicialización.

Por lo tanto, una vez devuelto el control del procesador al proceso principal y sabiendo cuál es el origen del movimiento, el siguiente paso es determinar cuál será la trayectoria que seguirá la silla para alcanzar el destino solicitado. Esta actividad, cuyo resultado es la decisión de la sucesión de nodos por los que irá pasando el móvil, es exactamente la misma que la desarrollada en [Marrón-00] y la descripción exacta de su funcionamiento puede encontrarse allí, ya que para los intereses de este trabajo es suficiente este enfoque de caja negra.

La siguiente inicialización corresponde a los parámetros que afectarán al EKF. La posición real del móvil pasará a ser la obtenida en la primera iteración de visión antes comentada y la matriz de covarianza del error de estimación, P, se inicializará como nula ya que partimos de información de visión y podemos considerarla como exacta.

5.2.2. Inicialización del proceso de odometría

Al igual que el subsistema de visión artificial, la lectura de datos procedentes de odometría se realizará en un proceso aparte y, por tanto, se inicializará como ya lo hicieron los otros.

Así, una vez pasa a ejecutarse este proceso con el mecanismo de dormir el proceso de mayor prioridad, sus primeras tareas consisten en la gestión de su planificación y prioridad como proceso, el bloqueo de las páginas de memoria asignadas y la indicación al sistema operativo de con qué funciones gestionará las señales que irá recibiendo del proceso principal. Este proceso se planificará como FIFO, igual que el principal, pero con una prioridad inferior a este en un nivel, como se explicará más adelante en el apartado dedicado a la planificación de las tareas.

En su parte de inicialización específica, el proceso se encarga de abrir, como sólo lectura, el fichero especial de dispositivo correspondiente al Neuron Chip que actúa de interfaz con la red de control de la silla de ruedas para obtener los datos de velocidad y tiempo acumulado que este le irá proporcionando. Asimismo, también tomará una referencia de en qué instante de tiempo se finalizó la inicialización, para utilizarla en posteriores cálculos.

Una vez preparado todo, se devuelve el control al proceso principal. Este proceso también abre el fichero especial de dispositivo correspondiente al Neuron Chip pero, en este caso, para escribir en él, ya que será el proceso principal el que se encargue de enviar a la red de control las consignas de velocidad que ha de adoptar cada rueda para seguir la trayectoria diseñada y alcanzar el destino. Después, ya sólo queda programar un temporizador que despierte al proceso principal cada 50 milisegundos (que, como ya se indicó, es el periodo de odometría y de la aplicación).

Llegada a este punto, la aplicación entra en un bucle en el que irá gestionando ya todo el movimiento y sólo se romperá al llegar a destino, momento en el que la silla se detendrá, se finalizarán los procesos secundarios (visión y odometría), se liberarán todos los recursos ocupados y terminará por completo la ejecución de la aplicación.

5.3. Planificación y comunicación entre módulos

Como se ha ido apuntando a lo largo de toda esta memoria, la planificación y la prioridad entre los procesos en juego durante el desarrollo de la aplicación tienen un papel fundamental en la consecución de los objetivos conseguidos. A lo largo de este apartado se va a intentar explicar lo más posible los mecanismos de planificación empleados y las decisiones de diseño adoptadas a este respecto a lo largo del proyecto.

5.3.1. Planificación en Linux

El sistema operativo elegido (Linux) implementa multitarea con tiempo compartido. Esto quiere decir que pueden existir simultaneamente varios procesos en ejecución y que el sistema operativo se encargará de dividir el

tiempo para que cada uno no se ejecute completo de una vez sino poco a poco, compartiendo ese tiempo de proceso. Así, se da al usuario la impresión de que se ejecutan a la vez (aunque aumente el tiempo requerido para su finalización) y este es el comportamiento de planificación por defecto que tiene este sistema operativo y muchos otros.

Sin embargo, este comportamiento no es el que más conviene a nuestra aplicación. Para cumplir los requisitos de diseño impuestos que garantizan la convergencia del EKF precisamos que el proceso principal, que incluye al EKF, y el de odometría se ejecuten cada 50 milisegundos y que, mientras se ejecuten, no haya otros procesos que tomen el procesador y alteren estos tiempos. Asimismo, el proceso que implementa el subsistema de visión artificial deberá aprovechar los tiempos muertos que dejan los otros dos procesos en esos 50 ms. de periodo de ejecución para ir procesando la información obtenida a través de la cámara de video y ofrecer datos de posición que fusionar con un periodo razonable.

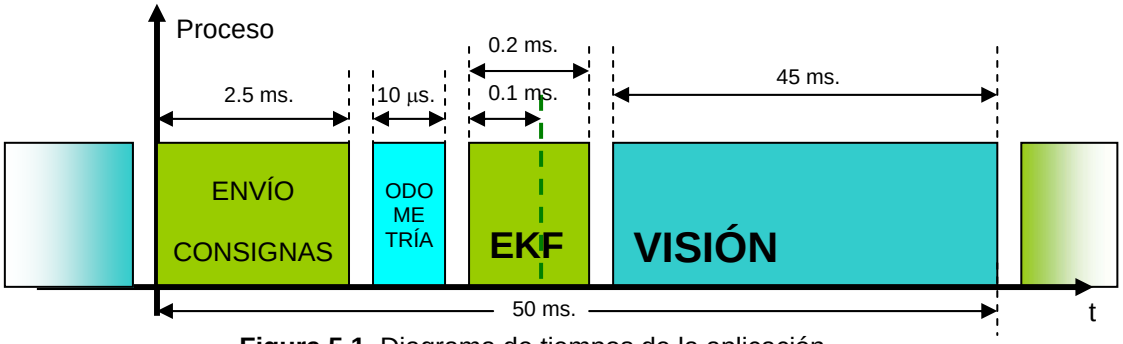


Figura 5.1. Diagrama de tiempos de la aplicación

Para conseguir que la aplicación se ajuste al comportamiento deseado, el sistema operativo proporciona otra política de planificación denominada `SCHED_FIFO` y que nos permite implementar procesos de pseudo-Tiempo Real. Así, los procesos para los que se sigue esta política de planificación están dotados con una prioridad que establece su orden de ejecución: en cualquier momento, de todos los procesos listos para ejecutarse, se ejecutará el de mayor prioridad y, además, se ejecutará sin interrupción hasta que finalice o hasta que un proceso de prioridad mayor pase a estado de listo para ejecutarse. Se dice que es pseudo-Tiempo Real porque este comportamiento es alterable por el kernel del sistema operativo.

Cuando el sistema operativo precisa ejecutar código del kernel lo hace sin atender ni prioridades ni interrupciones[17]. Además, el tiempo que puede durar esta situación no está acotado, depende del estado del sistema en ese momento (procesos en ejecución, tareas pendientes, etc). Sin embargo, como ya se indicó en el apartado de esta memoria dedicado a las consideraciones sobre Tiempo Real, se puede aproximar que el tiempo de 'interrupción' del kernel será lo suficientemente com bajo para no ser considerado si la carga del sistema es baja. En nuestro caso, como será un sistema dedicado en el que la

[17] Si se utilizase un auténtico núcleo de Tiempo Real (como, por ejemplo, RTLinux), el programa debería ejecutarse como un hilo del núcleo para hacer uso de planificación de Tiempo Real y prioridades.

única tarea de usuario será nuestra aplicación, nos ajustaremos a este supuesto.

Por lo tanto, el proceso principal así como el que controla las medidas de odometría estarán regidos por una planificación FIFO. Para garantizar el orden de ejecución, el proceso principal tomará la máxima prioridad posible para un proceso y el de odometría la inmediatamente inferior. Por otro lado, para el proceso de visión será suficiente con la política de planificación por defecto (`SCHED_OTHER`). Así este proceso aprovechará los huecod de ejecución que dejen las tareas más prioritarias para ir completando su misión. Una vez haya acabado, comunicará los resultados al proceso principal mediante un mensaje.

Para finalizar este apartado queda explicar cómo se ha implementado el cambio de una tarea a otra, aunque ya se hayan avanzado algunas cosas en anteriores apartados. Se utilizan dos mecanismos:

- Los procesos de mayor prioridad utilizan el mecanismo de dormirse (mediante la función `sleep()`) para ceder el procesador. De esta forma, le indican al sistema operativo que no quieren ejecutarse hasta que pase el tiempo indicado o sean interrumpidos, por ejemplo, por una señal.

- El otro mecanismo clave son las señales. Su modo de empleo es sencillo: cada proceso registra una función vacia (que no hace nada) como manejadora de una señal y, así, al recibir esta señal se interrumpirá el sueño, se ejecutará el manejador y se continuará a partir la siguiente instrucción a la que ordenó dormir al proceso. Por lo tanto, cuando un proceso quiera devolver el procesador a uno de mayor prioridad sólo tendrá que mandarle la señal correspondiente.

Para asegurar la ejecución del proceso principal (y, por tanto, de las lecturas de información de odometría y de la fusíon con el EKF) cada 50 ms. empleamos un temporizador. Esto sirve para que el sistema operativo envíe una señal al proceso principal cada vez que transcurra un tiempo dado y, de esta forma, aseguramos que vuelva a comenzar el ciclo de ejecución. Otra forma de ceder el procesador que también se utiliza en la aplicación consiste en que el proceso de mayor prioridad se quede bloqueado esperando al resultado de una entrada-salida.

Esto funciona de la manera siguiente:

- La ejecución del proceso principal no puede continuar si no se resuelve la operación de entrada-salida (lectura), por lo que este proceso queda en estado de espera.

- La resolución de la entrada-salida depende de que el proceso de menor prioridad escriba el dato que tiene que leer el proceso principal. El proceso de menor prioridad pasa a ejecutarse porque el otro está en espera.

Se esta forma se obtienen igualmente los resultados de planificación deseados. Este caso se explica con detalle en el apartado siguiente al ver el mecanismo de paso de resultados entre módulos mediante mensajes.

5.3.2. Comunicación entre módulos

Se han explicado los pormenores de la planificación modular de la aplicación, pero aún falta por ver cómo se van a comunicar los resultados obtenidos entre unos y otros.

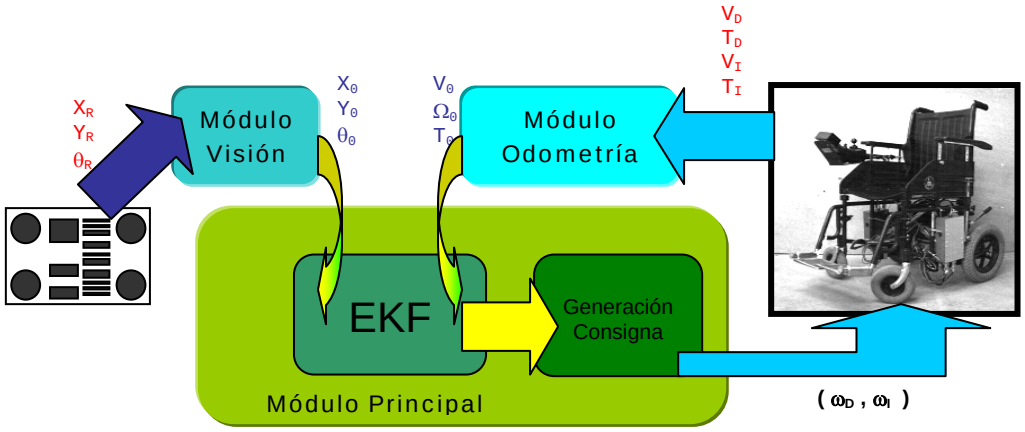


Figura 5.2. Esquema de la aplicación y paso de mensajes

Todos los procesos de los que consta la aplicación tienen un objetivo común que es conocer la posición exacta en la que se encuentra la silla a lo largo de su movimiento. Los procesos secundarios (odometría y visión) calculan esta posición y se la comunican al proceso principal. Después será éste el que, con ambas informaciones, estime cuál es la verdadera posición mediante el EKF y utilice este dato para llevar el movimiento a buen término.

Para este intercambio de datos entre procesos se ha utilizado el mecanismo conocido en teoría de sistemas operativos como colas de mensajes. En un principio se pensó en utilizar las definidas dentro del estándar POSIX, ya que en todo el proyecto se ha intentado utilizar las implementaciones de conceptos teóricos de sistemas operativos recogidos en este estandar. Desafortunadamente, Linux no implementa aún[18] las colas de mensajes POSIX, así que se decidió emplear la versión definida en el estandar de Unix System V (1983).

Por lo tanto, para dar solución a la comunicación entre procesos se creará una cola de mensajes que compartirán los tres. Además, se han definido tres tipos de mensajes para intercambiar datos útiles entre los procesos:

- ✓ Odometría: A través de este mensaje, el proceso de odometría comunica al procesoprincipál la velocidad lineal de la silla, su velocidad angular y el tiempo transcurrido desde la última medida.

[18] Está previsto que lo haga en futuras versiones.

- ✓ Visión: A través de este mensaje, el proceso de visión comunica al proceso principál la posición absoluta de la silla (coordenadas X, Y y θ) y el número de nodo que leyó en la marca artificial procesada.

- ✓ Mapa: A través de este mensaje, el proceso principal comunica al de visión que fichero de mapa debe utilizar para obtener la posición absoluta del nodo detectado (marca). Esto es útil para escenarios de movimiento que contemplen un cambio de edificio. En este proyecto aún no está implementada esa situación, aunque está previsto que se aborde en un futuro.

Otra de las posibilidades que ofrecen estas colas de mensajes es que se pueden realizar operaciones de envío y recepción de mensajes con caracter bloqueante o no bloqueante, esto es, cediendo el procesador hasta que se obtenga una respuesta (recepción o envío correcto) o continuar aunque no se reciba nada.

Ésto es útil, por ejemplo, para la comunicación del proceso principal con el proceso de visión: en cada iteración, el principal simplemente sondea para ver si el proceso visión envió datos y, después, actuará en consecuencia. Esto no puede ser así con la recepción de los datos de posición que envíe el proceso de odometría: sin ellos no se puede continuar y, por tanto, hasta su recepción se bloquea el proceso principal.

5.4. Funcionamiento normal de la Aplicación

Una vez completada la inicialización de la aplicación se debe haber alcanzado el siguiente punto:

- ✓ Los tres procesos de la aplicación (principal, odometría y visión) se están ejecutando.

- ✓ La planificación de los procesos y sus prioridades están ajustados a lo explicado anteriormente.

- ✓ Existe una cola de mensajes para la aplicación y los tres procesos tienen acceso a ella.

- ✓ El proceso principal y el de visión tienen acceso al fichero del mapa del edificio.

- ✓ El proceso principal y el de odometría tienen comunicación con el neuron.

- ✓ El proceso de visión tiene acceso a la cámara de video y ésta está correctamente configurada.

- ✓ Se conocen los nodos origen y destino del movimiento y la posición inicial, y con ello se ha diseñado una ruta para cubrir el trayecto.

Con todo esto listo, los tres procesos de los que consta la aplicación entran en un bucle que no se romperá hasta que se alcance el nodo de destino del movimiento. Ya se comentó que el periodo de la aplicación son 50 milisegundos, y ese será el periodo de ejecución de este bucle en el proceso principal y en el de odometría.

Como la aplicación conoce dónde se encuentra el robot en ese momento, lo primero que hace es enviar una consigna de movimiento a la silla de ruedas para que comience a desplazarse hacia el destino. De nuevo, el mecanismo es el mismo que se utilizaba en el trabajo de [Marrón-00]. Después se cede el turno al proceso de odometría para que nos dé la información que tiene de la velocidad (lineal y angular) que lleva la silla y el tiempo transcurrido desde la última medida. Esta información, además, se almacena para corregir después con ella el tiempo de proceso de los datos de visión artificial. Ésto se explicará algo más adelante.

Hay que tener en cuenta también que, en caso de que exista algún fallo, la información que nos llegue del proceso de odometría puede no ser correcta. Las causas de esto pueden ser la pérdida de algún paquete en las comunicaciones entre los Neuron de control de bajo nivel de la silla, por una pérdida de sincronismo entre la comunicación PC-Neuron, etc. En estos casos, como la información es imprescindible, se entiende que la velocidad que lleva la silla es la misma que llevaba la iteración anterior. Esta aproximación es válida porque la velocidad no habrá cambiado mucho de un periodo a otro.

La información de posición se envía al EKF, que puede funcionar de las siguientes formas:

- ✓ Si en la presente iteración hay información de posición obtenida mediante visión artificial se ejecutará el filtro completo, con sus fases de predicción y corrección. Con esto conseguimos que el dato de la posición actual sea más fiable, fusionando ambas informaciones.

- ✓ Si no hay datos de visión, únicamente se ejecutará la fase de predicción. En este caso, la información acerca de la posición del móvil es la misma que hubiéramos obtenido si el sistema funcionase únicamente basado en odometría, sin visión ni fusión.

Esto permite renovar la información que teníamos sobre la posición del móvil y, con ella, se calculará la nueva consigna a enviar a la silla para continuar el movimiento. El cálculo y el envío, como ya se dijo, constituyen el primer paso del ciclo.

En una iteración normal del bucle, el resto de los 50 ms del periodo de ejecución se dedicarían a que el subsistema de visión siga procesando la imagen adquirida. Sin embargo, existen iteraciones especiales del bucle en las que la ejecución se desvía un poco del esquema general que se ha presentado.

5.4.1. Iteraciones especiales.

Existen determinados momentos en los que se realizan más acciones que las comentadas hasta ahora. Estos vienen marcados por cuando se finaliza una iteración de visión y por la primera iteración en el cumplimiento de un subtrayecto (cada trayecto (ruta) se divide en subtrayectos entre nodo y nodo, y estos se dividen en tareas de avance y giro).

En la primera iteración del bucle que corresponda al cumplimiento de uno de los subtrayectos del movimiento, se aprovecha para planificar el siguiente. Este comportamiento, está justificado en el contexto del proyecto original de navegación sólo con información odometrica ([Marrón-00]).

Por otra parte, el hecho de que exista información de visión disponible al final de una iteración del bucle condiciona también la ejecución del siguiente. En el esquema general de funcionamiento se va almacenando la información que se recibe de odometría en cada iteración para luego corregir el tiempo de visión. Esto es necesario porque el subsistema de visión ofrece información de posición obtenida de una imagen capturada y que se ha tardado un tiempo grande (desde la perspectiva del movimiento) en procesar: tenemos información de una posición en la que estuvimos, pero en la que ya no estamos. La solución adoptada para corregir éste problema es añadir los avances detectados por odometría desde la iteración en la que se hizo la captura de imagen. Esta corrección se hace antes de llamar al EKF.

5.4.2. Finalización del programa.

Cuando se ha alcanzado el destino indicado y la silla se ha detenido la aplicación está lista para finalizar. Es en este momento en el que se envían las correspondientes señales a los procesos de odometría y visión para que liberen los recursos adquiridos: páginas de memoria asignadas, cola de mensajes y ficheros abiertos y hardware en uso (comunicación vía puerto paralelo con el neuron y tarjeta de TV). Después, el proceso principal liberará también sus recursos y la aplicación habrá finalizado.

6. Descripción de la Interfaz de Programación y Modelo de datos

En este capítulo se describen los tipos de datos y estructuras definidos en la aplicación diseñada y su significado dentro de ella, para que sirva de referencia en futuros desarrollos sobre el código implementado en este trabajo.

Después se pasa a describir la librería de funciones para trabajo con matrices utilizada en esta aplicación, y que se utiliza en el proceso de visión y en la implementación del EKF.

6.1. Modelo de datos de la aplicación

Posición

```
typedef struct
{
      float       x;
      float       y;
      float       Theta;
}Posicion;
```

Esta estructura se usa, a lo largo de la aplicación, para almacenar posiciones y orientaciones respecto del mapa del entorno.

Significado de sus miembros:

- **x**: Corresponde a la coordenada ‘x’ en el mapa del entorno, expresada en centímetros a escala 1:500.

- **y**: Corresponde a la coordenada 'y' en el mapa del entorno, expresada en centímetros a escala 1:500.
- **Theta**: Corresponde al ángulo formado por la orientación del objeto de posicionamiento tratado y el eje 'x' del mapa del entorno, expresado en grados. Por ejemplo: si en esta estructura está almacenada la posición de la silla, sería el ángulo formado por el vector que expresaría el sentido de su velocidad lineal (orientación del móvil) y el eje 'x'.

Giro

```
typedef struct
{
      float       xCentro;
      float       yCentro;
      float       Radio;
}Giro;
```

En esta estructura se almacenan los datos necesarios para una tarea de giro planificada dentro de un trayecto, en el contexto del módulo principal de navegación y control . Al igual que la posición, las unidades utilizadas son centimetros, con escala 1:500.

Significado de sus miembros:

- **xCentro**: Coordenada 'x' del centro de la circunferencia que contiene el arco de giro deseado.
- **yCentro**: Coordenada 'y' del centro de la circunferencia que contiene el arco de giro deseado.
- **Radio**: Radio de la circunferencia que contiene el arco de giro deseado.

Trayectoria

```
typedef struct
{
      Posicion    Origen;
      Posicion    Destino;
      Giro        Circulo;
      int         Status;
}Trayectoria;
```

Esta estructura sirve para almacenar los datos necesarios de las sub-tareas en las que de divide la ruta entre el nodo origen y el nodo destino del trayecto completo, en el contexto del módulo principal de navegación y control. En ellas se almacenarán los datos necesarios para cada tarea de avance o de giro.

Significado de sus miembros:

- **Origen**: Estructura de tipo posición que contiene las coordenadas de la posición origen del movimiento que implica la tarea y el ángulo de orientación esperado de inicio del movimiento.

- **Destino**: Estructura de tipo posición que contiene las coordenadas de la posición destino del movimiento que implica la tarea y el ángulo de orientación esperado de finalización del movimiento.
- **Circulo**: Estructura con los parámetros necesarios para el giro, si la tarea es de este tipo.
- **Status**: Aquí se almacenará el tipo de tarea de la que se trata:
    - 1 = Avance.
    - 2 = Giro.

tMatriz

```
typedef struct
{
      int         fil, col;
      int         *fil_real, *col_real;
      float       **datos;
}tMatriz;
```

Esta estructura es la base de la librería para trabajo con matrices libmatrices, definida y descrita en este proyecto. Con ella se define una matriz de n filas y m columnas a cuyos datos se accede mediante el miembro `datos[m][n]`. Esta estructura debe ser inicializada mediante la función CrearMatriz (m,n) y debe ser liberada cuando ya no se necesite mediante la función LiberarMatriz(), ambas incluidas en libmatrices. Esto es debido a que hacen uso de asignación dinámica de memoria.

Significado de sus miembros:

- **fil**: Número máximo de filas que puede tener una matriz contenida en esta estructura, esto es, el número máximo de filas para las que se ha asignado memoria en esta estructura. Este valor se establece al llamar a CrearMatriz(), y es fijo desde ese momento hasta su liberación.
- **col**: Número máximo de columnas que puede tener una matriz contenida en esta estructura, esto es, el número máximo de columnas para las que se ha asignado memoria en esta estructura. Este valor se establece al llamar a CrearMatriz(), y es fijo desde ese momento hasta su liberación.
- **fil_real**: Número de filas real que tiene la matriz contenida en la estructura en un momento dado. Por ejemplo: podemos haber definido espacio para una matriz de 300x200 (dimensiones máximas fil x col), pero almacenar en la estructura una de 6 x 4 (dimensiones reales fil_real x col_real). Este valor no es fijo, se puede ver afectado, por ejemplo, al almacenar en una tMatriz el resultado de una operación matricial (de las definidas en libmatrices).
- **col_real**: Número de columnas real que tiene la matriz contenida en la estructura en un momento dado. Por ejemplo: podemos haber definido espacio para una matriz de 300x200 (dimensiones máximas fil x col), pero almacenar en la estructura una de 6 x 4 (dimensiones reales fil_real x col_real). Este valor no es fijo, se puede ver afectado, por ejemplo, al almacenar en una tMatriz el resultado de una operación matricial (de las definidas en libmatrices).

- **datos**: Puntero a puntero a dato en coma flotante que sirve para acceder a los datos de la matriz como si de un array de dos dimensiones se tratara.

buf_msg_vi

```
typedef struct
{
      long        mtype;
      char        nodo[6];
      float       datos[3];
}buf_msg_vi;
```

Estructura definida para el paso de mensajes desde el módulo de visión al módulo principal de esta aplicación.

Significado de sus miembros:

- **mtype**: Tipo de mensaje, será siempre 2 para ser identificado por el módulo destino (principal) como un mensaje procedente del módulo de visión.
- **nodo**: Nombre del nodo correspondiente a la marca analizada por el módulo de visión.
- **datos**: Array de valores en coma flotante, en cuyas posiciones se almacena:
    - Posición 0: Coordenada x de la posición absoluta del móvil según el subsistema de visión, en centímetros con escala 1:500.
    - Posición 1: Coordenada y de la posición absoluta del móvil según el subsistema de visión, en centímetros con escala 1:500.
    - Posición 2: Ángulo θ de orientación absoluta del móvil según el subsistema de visión, expresada en grados.

buf_msg_od

```
typedef struct
{
        long      mtype;
        float     datos[3];
}buf_msg_od;
```

Estructura definida para el paso de mensajes desde el módulo de odometría al módulo principal de esta aplicación.

Significado de sus miembros:

- **mtype**: Tipo de mensaje, será siempre 1 para ser identificado por el módulo destino (principal) como un mensaje procedente del módulo de odometría.

- **datos**: Array de valores en coma flotante, en cuyas posiciones se almacena:

    - Posición 0: Velocidad lineal del móvil, en m/s.
    - Posición 1: Velocidad angular del móvil, en rad/s.
    - Posición 2: Tiempo transcurrido desde la última medida, en segundos.

buf_msg_map

```
typedef struct
{
        long      mtype;
        char      edificio;
}buf_msg_map;
```

Estructura definida para el paso de mensajes desde el módulo principal a otro módulo de esta aplicación para que conozca con qué mapa debe trabajar.

Significado de sus miembros:

- **mtype**: Tipo de mensaje, será siempre 1 para ser identificado como un mensaje de este tipo.
- **edificio**: Contendrá una letra que identifique al edificio: 'O' será oeste, 'E' será este, 'N' será norte y 'S' será sur.

paquete_n_pc

```
typedef union
{
      char                    contenido[TAM_PAQUETE_N_PC];
      struct
      {
            unsigned short    ta_dcha;
            short             va_dcha;
            unsigned short    ta_izda;
            short             va_izda;
      } n_pc;
} paquete_n_pc;
```

Estructura definida para almacenar y tratar la información recibida del Neuron Chip.

Significado de sus miembros:

- **contenido**: Se utiliza como puntero a cadena de caracteres, y apunta al comienzo de la estructura. Es útil para utilizar como parámetro en las órdenes de lectura en lugar de ' &[nombre_de_la_estructura]'. Se define por comodidad y claridad en el código.

- **ta_dcha**: Tiempo acumulado en el que se realizó la medida de velocidad de la rueda derecha, expresado en ticks. Un tick son 0.82 milisegundos

- **va_dcha**: Número de pulsos acumulados en la rueda derecha. Según cuantos haya habido desde la medida anterior se calcula la distancia recorrida gracias al movimiento de esta rueda.
- **ta_izda**: Tiempo acumulado en el que se realizó la medida de velocidad de la rueda izquierda, expresado en ticks. Un tick son 0.82 milisegundos
- **va_izda**: Número de pulsos acumulados en la rueda izquierda. Según cuantos haya habido desde la medida anterior se calcula la distancia recorrida gracias al movimiento de esta rueda.

paquete_pc_n

```
typedef union
{
      char                    contenido[TAM_PAQUETE_PC_N];
      struct
      {
            char              comando;
            char              longitud;
            short             v_dcha;
            short             v_izda;
            char              eom;
      } pc_n;
} paquete_pc_n;
```

Estructura definida para enviar las consignas necesarias al control de bajo nivel de la silla.

Significado de sus miembros:

- **contenido**: Se utiliza como puntero a cadena de caracteres, y apunta al comienzo de la estructura. Es útil para utilizar como parámetro en las órdenes de escritura en lugar de ' &[nombre_de_la_estructura]'. Se define por comodidad y claridad en el código.

- **comando**: Tal y como está implementado, el driver de comunicación con el neuron necesita que le indiquemos el comando del protocolo de comunicación que estamos escribiendole situado al principio del paquete que le enviamos. Por lo tanto, el valor debe ser 0x01, correspondiente a una escritura.
- **longitud**: longitud del paquete, en bytes, contando a partir del siguiente campo del paquete e ignorando el campo 'eom'. Será siempre 4 bytes.
- **v_dcha**: Velocidad que queremos que lleve la rueda derecha, en mrad/s.
- **v_izda**: Velocidad que queremos que lleve la rueda izquierda, en mrad/s.
- **eom**: Indica el final del paquete, será siempre 0x00.

6.2. Descripción del API matricial

Aquí se explicarán las funciones definidas en la librería 'libmatrices' diseñada para facilitar el trabajo con matrices y que está profundamente basada en el tipo de datos 'tMatriz', definido en el apartado anterior.

El hecho de que los datos relevantes del tipo tMatriz sean gestionados a través de punteros hace que, aunque pasemos argumentos por valor de tipo tMatriz a una función, sí modifiquemos el valor real almacenado (como si hubiéramos pasado el argumento por referencia) ya que accedemos a él a través del puntero. Este enfoque se utiliza en todas las funciones en las que el resultado es una matriz, por lo que este resultado se devolverá en alguna de las matrices pasadas como parámetro.

CrearMatriz

```
tMatriz CrearMatriz(int N_Fil,int N_Col)
```

Permite la creacion dinámica de matrices. Asignamos memoria a la matriz del tipo estructurado tMatriz, ademas de incorporar la información del número de filas y columnas que contiene.

Parámetros de entrada:

- **N_Fil**: Indica el número de filas de la matriz a crear.
- **N_Col**: Indica el número de columnas de la matriz a crear.

Parámetros de salida:

- **matriz**: Devuelve la matriz con el tamaño y memoria asignada.

LiberarMatriz

```
void LiberarMatriz(tMatriz matriz)
```

Libera la memoria usada por la matriz.

Parámetros de entrada:

- **matriz**: Matriz la cual hay que liberar memoria.

Parámetros de salida: Ninguno.

TamanoMatriz

```
void TamanoMatriz(  int *N_Fila,
                    int *N_Col, tMatriz matriz)
```

Proporciona el numero de filas y columnas que tiene una matriz determinada. Aunque su uso pueda resultar 'excesivo', se utiliza por claridad y por comodidad a la hora de programar.

Parámetros de entrada:

- **N_Fila**: Pasado por referencia. Contiene el numero de filas.
- **N_Col**: Pasado por referencia. Contiene el numero de columnas.
- **matriz**: Es la matriz de la cual queremos obtener el numero de filas y columnas.

Parámetros de salida: Ninguno, son pasados por referencia.

CerosMatriz

```
void CerosMatriz(   int filas,
                    int columnas, tMatriz matriz)
```

Rellena una matriz con tantos ceros como (filas x columnas) se quieran.

Parámetros de entrada:

- **filas**: Número de filas que se quiere rellenar.
- **columnas**: Número de columnas que se quiere rellenar.
- **matriz**: Matriz a rellenar.

Parámetros de salida: Ninguno, son pasados por referencia

UnosMatriz

```
void UnosMatriz(    int filas,
                    int columnas, tMatriz matriz)
```

Rellena una matriz con tantos unos como (filas x columnas) se quieran.

Parámetros de entrada:

- **filas**: Número de filas que se quiere rellenar.
- **columnas**: Número de columnas que se quiere rellenar.
- **matriz**: Matriz a rellenar.

Parámetros de salida: Ninguno, son pasados por referencia

CreaIdentidad

```
void CreaIdentidad(tMatriz matriz)
```

Iguala la matriz pasada a la matriz identidad

Parámetros de entrada:

- **matriz**: Matriz a igualar a la identidad

Parámetros de salida: Ninguno, son pasados por referencia

Sum

```
void Sum(tMatriz x, tMatriz y, int indicador)
```

Suma los valores de cada columna o fila.

Parámetros de entrada:

- **x**: Matriz con los valores a sumar
- **y**: Matriz con el resultado de la suma de cada columna o fila.
- **indicador**: Si es 1 suma columnas, si es 2 suma filas

Parámetros de salida: Ninguno, son pasados por referencia

ClasificaFila

```
void ClasificaFila (tMatriz x, int n)
```

Ordena las filas segun indique el orden ascendente de la columna n

Parámetros de entrada:

- **x**: Matriz de filas a ordenar.
- **n**: Indica el número de columna a ordenar.

Parámetros de salida: Ninguno, son pasados por referencia.

Buscar

```
void Buscar(   tMatriz entrada, tMatriz salida,
               int indicador, float valref)
```

Busca los índices de los elementos que no sean ceros.

Parámetros de entrada:

- **entrada**: Matriz de elementos a procesar.
- **salida**: Matriz que contiene los indices de los elementos que no son ceros.
- **indicador**:
    - si es !, selecciona indices de elementos distintos de...
    - si es >, selecciona indices de elemnetos mayores de...
    - si es <, selecciona indices de elementos menores de ...
    - si es =, selecciona indices de elementos iguales a ...

- **valref**: Valor de referencia.

Parámetros de salida: Ninguno, son pasados por referencia.

Abs

```
void Abs(tMatriz matriz)
```

Calcula el valor absoluto de los elementos de una matriz dada.

Parámetros de entrada:

- **matriz**: Matriz a procesar.

Parámetros de salida: Ninguno, son pasados por referencia

Redondeo

```
float Redondeo(float valor)
```

Redondea al valor entero mas cercano.

Parámetros de entrada:

- **valor**: Número real a redondear.
- Parámetros de salida: Valor redondeado.

Max

```
float Max(tMatriz matriz)
```

Halla el elemento con mayor valor de una matriz.

Parámetros de entrada:

- **matriz**: Matriz de datos a procesar.

Parámetros de salida: Valor máximo de la matriz.

Min

```
float Min(tMatriz matriz)
```

Halla el elemento con menor valor de una matriz.

Parámetros de entrada:

- **matriz**: Matriz de datos a procesar.

Parámetros de salida:

- **res**: Valor mínimo de la matriz.

Diff

```
void Diff(tMatriz entrada, tMatriz salida)
```

Crea una matriz con (i-1) filas, compuestas del resultado de restar al elemento (i+1, j) el elemento (i , j) de la matriz de entrada.

Parámetros de entrada:

- **entrada**: Matriz de datos a procesar.
- **salida**: Matriz ya procesada.

Parámetros de salida: Ninguno, son pasados por referencia.

Media

```
void Media(    tMatriz entrada,
               int indicador, float* result)
```

Halla el valor medio de los elementos de un vector columna.

Parámetros de entrada:

- **entrada**: Vector de entrada.
- **indicador**:
  - Si es 1, halla el valor medio de cada columna.
  - Si es 2, halla el valor medio de cada fila.
- **result**: Valor medio resultante. Se pasa por referencia.

Parámetros de salida: Ninguno, son pasados por referencia.

MVS

```
float MVS(float vect[],int numelem)
```

Halla el máximo valor singular del vector de entrada.

Parámetros de entrada:

- **vect**: Vector de entrada.
- **numelem**: Número de elementos del vector.

Parámetros de salida: Resultado de la operacion.

Signo.

```
float Signo(float valor)
```

Comprueba si el valor es mayor, menor o igual a cero.

Parámetros de entrada:

- **valor**: Valor a comprobar.

Parámetros de salida:

  - Retorna 1 si valor>0.
  - Retorna -1 si valor<0.
  - Retorna 0 si valor=0.

Normal

```
void Normal(tMatriz entrada,tMatriz salida)
```

Cambia cualquier matriz X al rango [0,1], esto es, la normaliza. Para ello, toma el valor maximo de a y el minimo b y aplica la relacion: Y=((X-b)/(a-b))

Parámetros de entrada.

- **entrada**: Matriz con datos a normalizar.
- **salida**: Matriz con los datos normalizados.

Parámetros de salida: Ninguno, son pasados por referencia.

Polyfit.

```
void Polyfit(int X, tMatriz Y, float *v)
```

Obtiene el polinomio de segundo grado de mejor ajuste para el conjunto de datos dados mediante el calculo de minimos cuadrados:

$v=[c\ b\ a] \Rightarrow p(x)=a \cdot x^2 + b \cdot x + c$

$$A=\begin{bmatrix} 1 & X_1^1 & \dots & X_1^m \\ 1 & X_2^1 & \dots & X_2^m \\ . & . & . & . \\ . & . & . & . \\ . & . & . & . \\ 1 & X_n^1 & \dots & X_n^m \end{bmatrix} \qquad Y=[Y_1\ Y_2\ \dots\ Y_n]$$

$$v=(A^T \times A)^{-1} \times A^T \times Y$$

Parámetros de entrada:

- **X**: Número de puntos del eje X.
- **Y**: Vector Y.
- **v**: Vector v.

Parámetros de salida: Ninguno, son pasados por referencia.

Multiplica.

```
void Multiplica(    tMatriz result,
                    tMatriz A, tMatriz B)
```

Obtiene el resultado de multiplicar dos matrices de tamaño ilimitado.

Parámetros de entrada:

- **A**: Matriz de datos A.
- **B**: Matriz de datos B.
- **result**: Resultado de la multiplicación.

Parámetros de salida: Ninguno, son pasados por referencia

Suma.

```
void Suma(tMatriz result, tMatriz A, tMatriz B)
```

Obtiene el resultado de sumar dos matrices de tamaño ilimitado.

Parámetros de entrada:

- **A**: Matriz de datos A.
- **B**: Matriz de datos B.
- **result**: Resultado de la suma.

Parámetros de salida: Ninguno, son pasados por referencia

Resta.

```
void Resta(tMatriz result, tMatriz A, tMatriz B)
```

Obtiene el resultado de restar dos matrices de tamaño ilimitado.

Parámetros de entrada:

- **A**: Matriz de datos A.
- **B**: Matriz de datos B.
- **result**: Resultado de la resta.

Parámetros de salida: Ninguno, son pasados por referencia

Traspuesta.

```
void Traspuesta(tMatriz traspuesta,tMatriz A)
```

Obtiene la matriz traspuesta de una dada.

Parámetros de entrada:

- **A**: Matriz de datos A.
- **traspuesta**: Matriz traspuesta de A.

Parametros de salida: Ninguno, son pasados por referencia

Inversa.

```
void Inversa(tMatriz A, tMatriz Ainv)
```

Función que obtiene la inversa de una matriz, mediante el uso del metodo de Gauss-Jordan.

Parámetros de entrada:

- **A**: Matriz a invertir.
- **Ainv**: Matriz invertida.

Parámetros de salida: Ninguno, son pasados por referencia

7. Resultados, Conclusiones y Trabajos Futuros

Una vez desarrollado y expuesto el sistema completo solamente queda mostrar los resultados y conclusiones obtenidos del mismo. En este capítulo se muestran mediante gráficas realizadas a partir de datos obtenidos de ejecuciones en Tiempo Real, el resultado de aplicar al sistema de navegación la fusión de de datos de odometría y visión para la mejora del posicionamiento.

Además, a lo largo del trabajo se han ido apuntando diversas ideas para realizar en un futuro de cara a la mejora de la platadorma actual. Todas ellas se recopilan al final del presente capítulo.

7.1 Resultados obtenidos.

Para probar la plataforma diseñada se pensó en un escenario de movimiento lineal entre dos nodos situados en los extremos inicial y final del pasillo 340 del edificio oeste de la Escuela Politécnica de la Universidad de Alcalá.

Para obtener cómo sería una respuesta ideal del sistema y poder comparar así con la prueba real que se hizo después se realizó una ejecución de la aplicación sobre el mismo trayecto con las siguientes condiciones forzadas en el propio código de la aplicación:

- La silla de ruedas recibe las consignas pero no las aplica, sino que devuelve el mismo valor que se le envió. De esta forma simulamos un comportamiento ideal del robot, que ejecuta inmediatamente y sin error las consignas que se le envían.

- El subsistema de visión siempre es capaz de decodificar el código de barras de la marca y, con esa información, obtener la posición absoluta del móvil.

En esta simulación la silla de ruedas se encontraba quieta y con la cámara fija delante de una marca a aproximadamente un metro de distancia de ella. Con las condiciones impuestas, la aplicación evoluciona en función de la posición que va estimando a partir de lo que piensa que le devuelven los encoders de la silla, que no es más que la orden que la aplicación misma ha enviado antes.

Para simular la visión se calculó cuántas iteraciones de visión completas tardaría el móvil en llegar a la marca final y se impuso que la aplicación detectaría la marca inicial en la primera iteración y, pasado el número de iteraciones correspondiente sin detectar ninguna marca, alcanzaría la marca situada al final del pasillo.

Por otro lado, se realizó una prueba real de la plataforma diseñada en el mismo escenario de movimiento lineal antes comentado. Para ello, se colocaron sendas marcas en el quicio de las puertas que delimitan el acceso al pasillo (340-O) y al laboratorio del fondo del pasillo (349-O) y se situó a la silla a una distancia de aproximadamente un metro de la primera marca, de tal forma que pudiera ver el código de barras en ella impreso con un tamaño en la imagen suficiente para su decodificación.

7.1.1 Resultados de la simulación

En la gráfica siguiente se muestra una representación realizada con MATLAB de la posición instantánea estimada por la aplicación en cada periodo de ejecución (50 ms).

En ella se puede observar como sigue perfectamente la ruta en linea recta a lo largo del pasillo y finaliza el movimiento en el punto esperado. En la gráfica, las coordenadas X e Y corresponden al Sistema de Coordenadas General (SCG) explicado ya en esta memoria, y están medidas en centímetros escalados 1:500 al igual que en el mapa de la aplicación. En este mapa, las coordenadas de los nodos origen y destino son las siguientes:

Nodo	Coord. X	Coord. Y	Theta
340-O	7.00	5.34	180º
349-O	11.10	5.34	180º

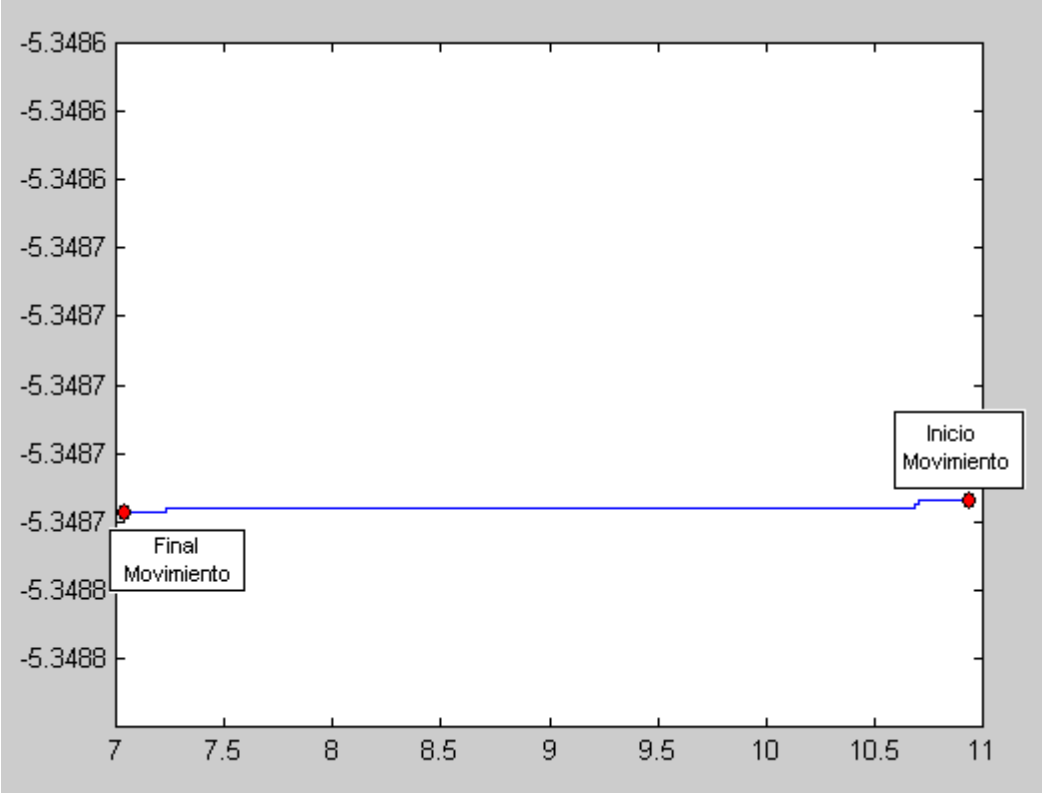


Figura 7.1. Resultados de la simulación de una trayectoria recta

Por otro lado, en las siguientes figuras se muestran las consignas de control enviadas:

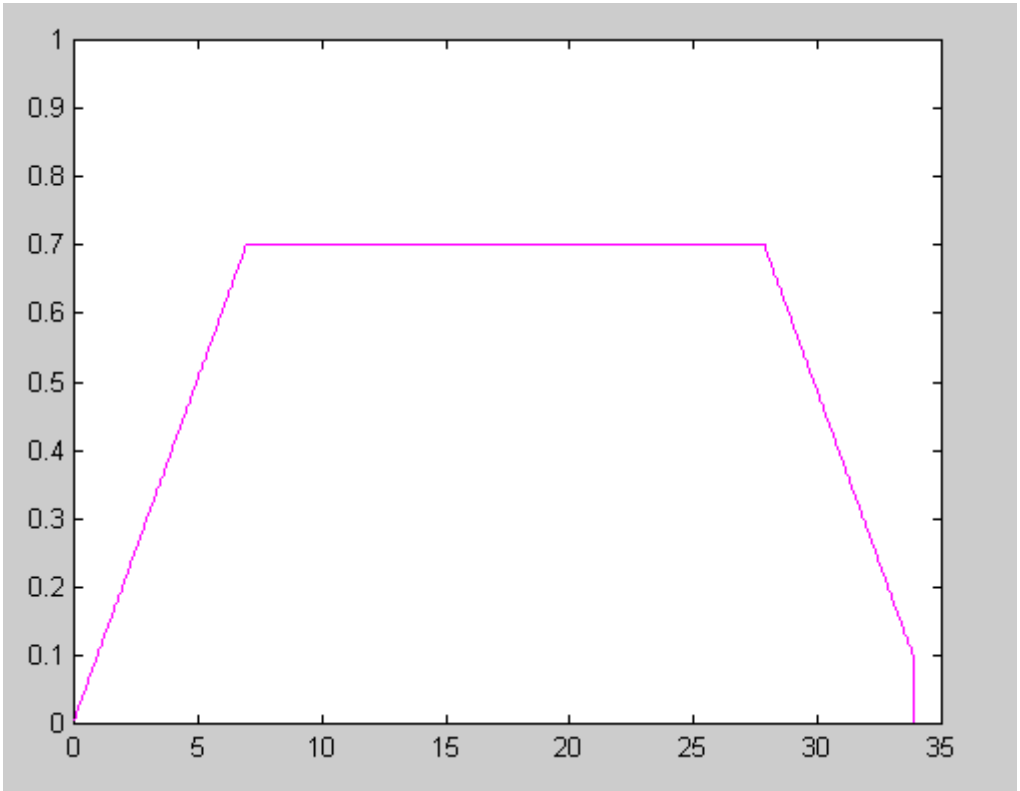


Figura 7.2. Consigna de velocidad lineal (m/s) en la simulación de una trayectoria recta

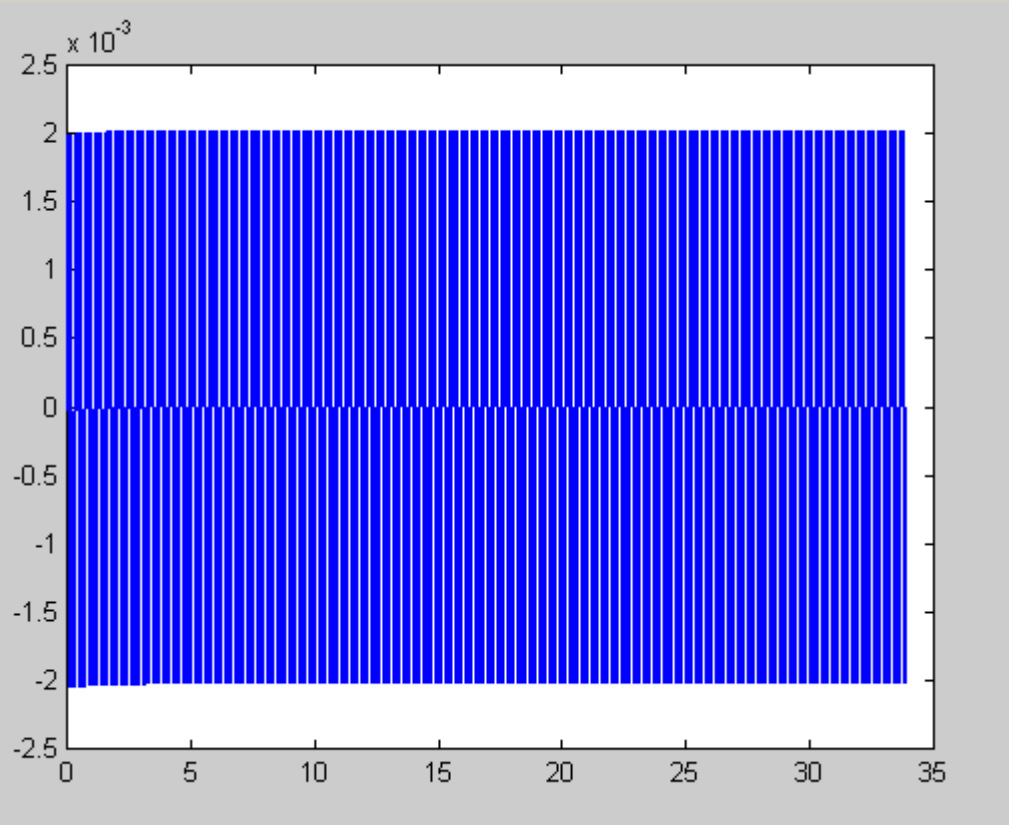


Figura 7.3. Consigna de velocidad angular (rad/s) en la simulación de una trayectoria recta

En estas figuras se observa que la consigna de velocidad lineal sigue un perfil trapezoidal y que la consigna de velocidad angular oscila alrededor de un valor nulo, si bien el valor de esta oscilación es muy pequeño.

7.1.2 Resultados de la prueba real

A continuación se exponen los resultados obtenidos en el escenario de pruebas descrito más arriba. En este caso, al realizar una prueba real, los resultados se separan del comportamiento ideal simulado, como se puede observar en las gráficas.

En la primera de ellas se presentan los resultados de posicionamiento a lo largo de la ruta recorrida en color azul. Además, en la misma gráfica se ha añadido una línea recta en color verde que representa cuál debería haber sido la trayectoria recorrida por el robot. Como se puede ver, el movimiento real oscila alrededor de la línea recta, si bien son diferencias pequeñas.

En las dos figuras siguientes, 7.4 y 7.5, se representan las consignas de velocidad lineal y angular enviadas al robot durante el movimiento. Al igual que en la otra figura, se observa un comportamiento que difiere del ideal aunque dentro de lo esperado. La consignas de velocidad lineal mantienen su perfil tra-

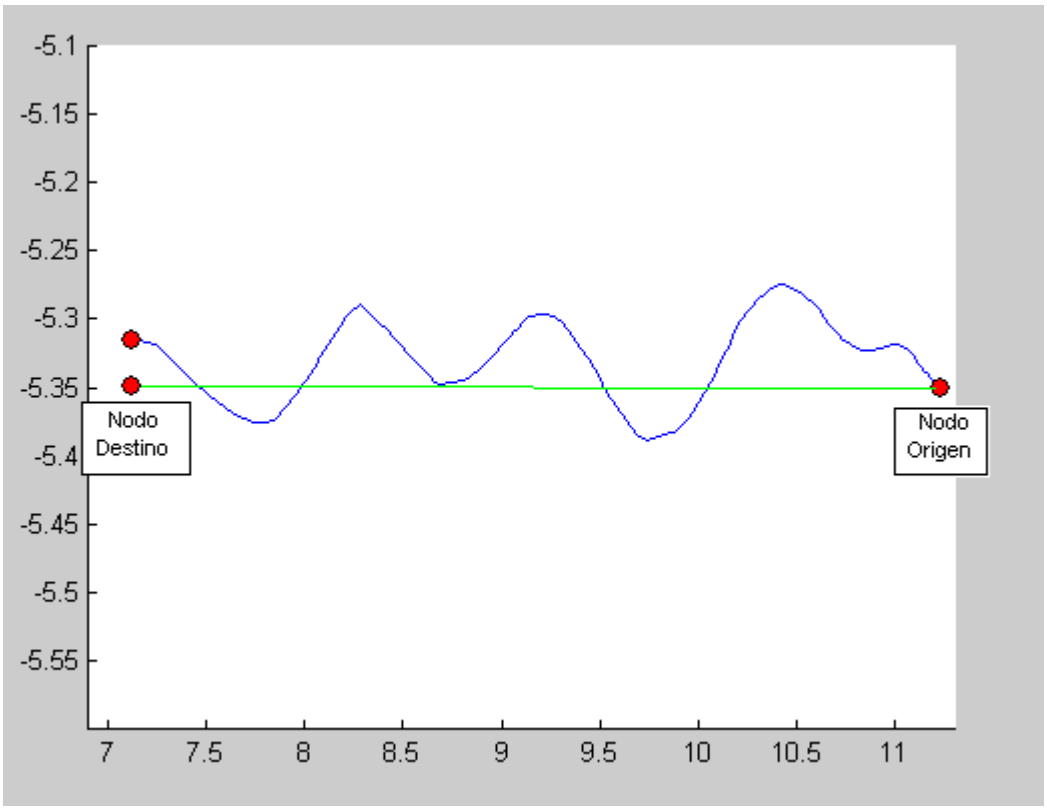


Figura 7.3. Resultados de posición en la prueba (cms/500).
Trazo azul: Trayectoria seguida por el móvil
Trazo verde: Trayectoria ideal en línea recta.

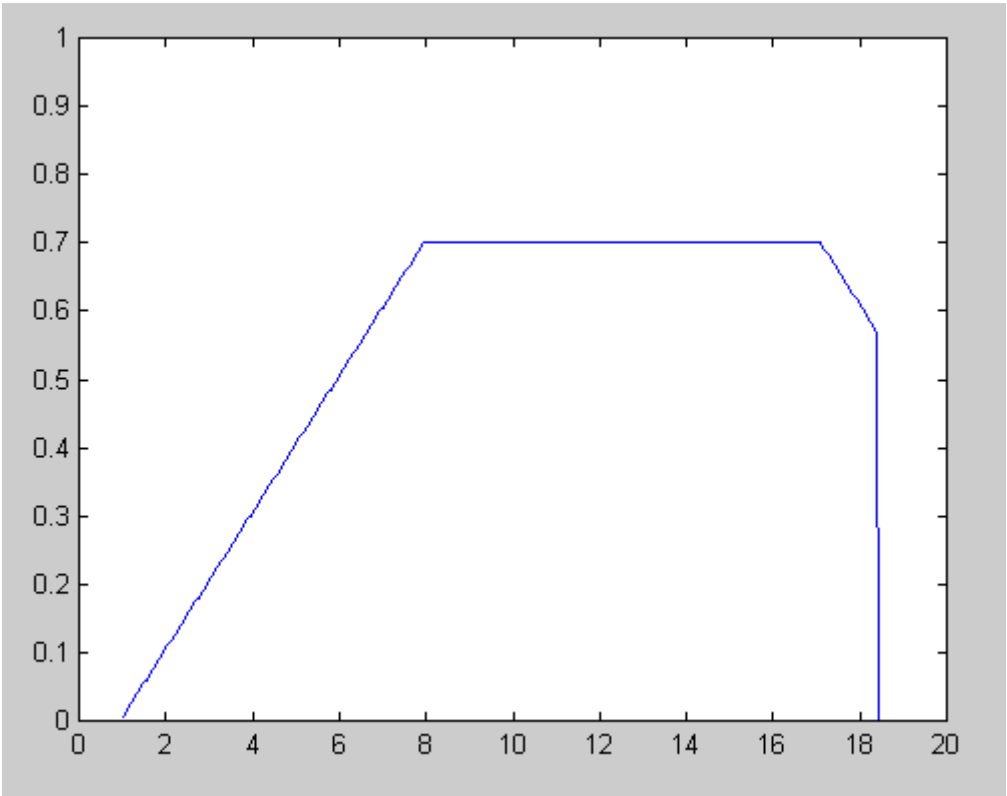


Figura 7.4. Consignas de velocidad lineal en la prueba (m/s).

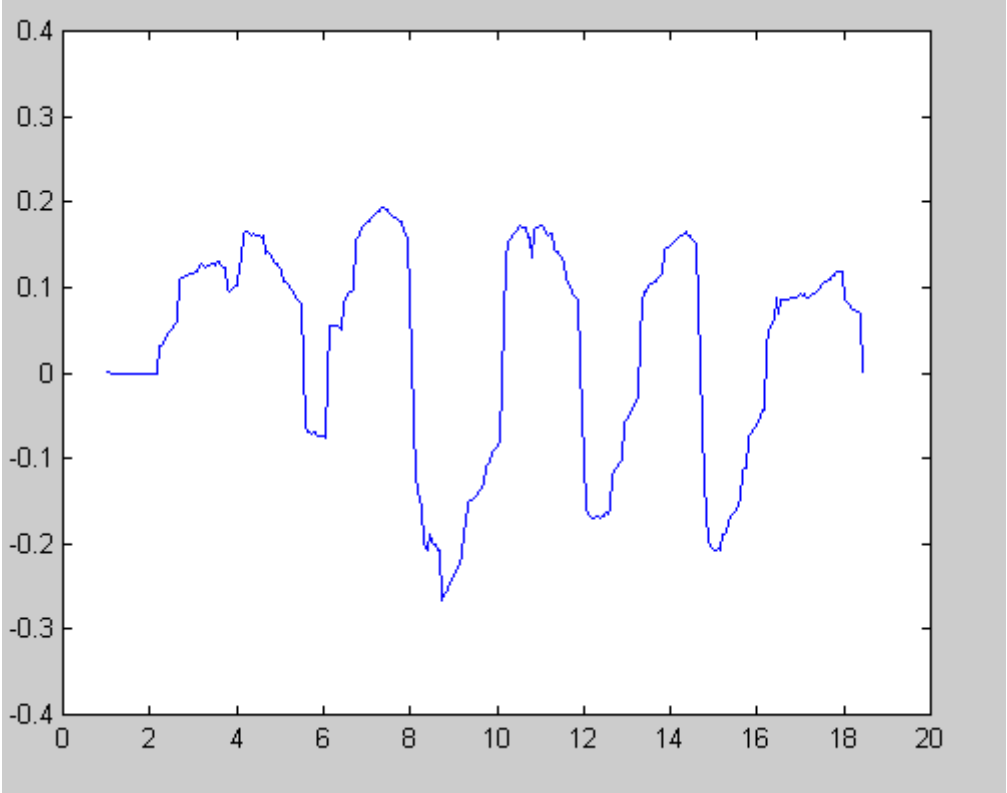


Figura 7.5. Consignas de velocidad angular en la prueba (rad/s).

pezoidal, pero las consignas de velocidad angular presentan una oscilación mucho más acusada alrededor de la consigna nula. Esto es debido a que el controlador intenta compensar la respuesta ligeramente desigual entre la respuesta de ambas ruedas ante las consignas.

Por último, en las figuras 7.6 y 7.7 se muestra el error entre las consignas de velocidad lineal y angular y la respuesta real de la silla, donde se observa de nuevo la oscilación en torno al error nulo.

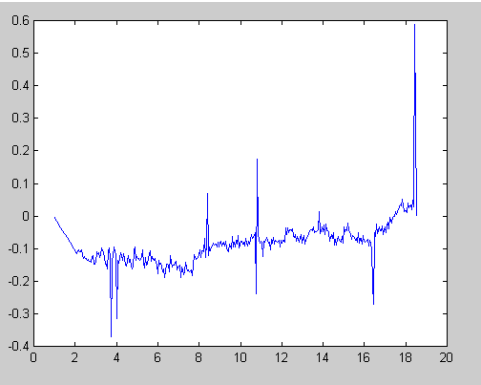


Figura 7.6. Error de velocidad lineal en la prueba (m/s).

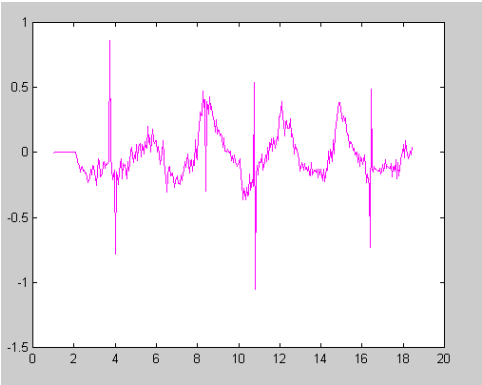


Figura 7.7. Error de velocidad angular en la prueba (rad/s).

7.2. Conclusiones y Trabajos Futuros

A lo largo de este trabajo se han señalado algunas mejoras que deben ser acometidas para que la aplicación alcance un funcionamiento más cercano al esperado.

Sin duda, un trabajo que sería importante realizar a partir de este proyecto es su adaptación para ser ejecutado sobre un sistema operativo de tiempo real como, por ejemplo, RTLinux. Aunque las suposiciones de cercanía a la ejecución en tiempo real realizadas en este trabajo se cumplen en la mayor parte del tiempo de ejecución de la aplicación, como ya se comentaba en el capítulo dedicado a la integración de trabajos ésto es dependiente de la carga del sistema y de las tareas que, debido a esa carga, tenga que ejecutar el núcleo del sistema operativo cuando tome el procesador. Como el tiempo que necesita el kernel no esta acotado y no éste no es interrumpible, una situación como la descrita puede alterar puntualmente el esquema de temporización de la aplicación. Este problema se evitaría utilizando un núcleo de tiempo real, lo cual implicaría:

- Modificar los procesos de la aplicación para que se ejecuten no como programas de usuario sino como módulos cargados en el núcleo del sistema operativo.
- Reescribir el sistema de comunicaciones entre procesos.
- Modificar la gestión de prioridades entre los procesos para que se ajusten a la nueva plataforma.

Otro posible trabajo consiste en sustituir la cámara de video analógica empleada por una digital. De esta forma se podría trabajar diréctamente con los datos de la cámara y se aumentaría la velocidad de adquisición y procesado evitando el driver analógico.

También se podría añadir un sistema de *tracking* que buscase la marca en la imagen capturada en función de dónde estuviera ubicada ésta en la iteración anterior, y utilizar mecanismos de ampliación de la imagen y un pan-tilt controlado por la aplicación que ayudase a realizar detecciones de marca más rápidas y eficaces.

Otra mejora necesaria respecto al subsistema de visión pasa por optimizar la implementación del algoritmo de procesado de imágenes capturadas para reducir su tiempo de ejecución.

Por otro lado, otro trabajo podría ocuparse de que la aplicación tenga un mayor conocimiento previo de las marcas que ve en su camino sin depender de la información del código de barras. Sería posible implementar un código que determinase cuál es la marca presente en una imagen capturada en función de la ruta diseñada por la aplicación y el trayecto ya recorrido.

También se podría realizar, de alguna forma, una distinción entre las marcas a la entrada y salida de un mismo nodo (por ejemplo, en el lado interior y exterior de la puerta de una sala). Serían soluciones válidas duplicar el

número de marcas, haciendo que sean marcas distintas, o indicar esta circunstancia de alguna forma (como, por ejemplo, la inversión del código Barker).

III. Manual del 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 todos los sistemas físicos y lógicos descritos en el apartado de 'Pliego de Condiciones': la plataforma completa que se mostró en el capítulo 3 de la memoria (silla de ruedas con su electrónica de bajo nivel), un PC sobre el que se ejecute la aplicación, cuyo puerto paralelo se comunicará con la electrónica de bajo nivel de la silla y dotado con tarjeta de TV y una cámara de video.

III.1. Plataforma PC

El PC donde va a correr la aplicación precisa de un Sistema Operativo Linux con la versión del kernel apropiada (2.4.22) y deberá tener montada una tarjeta de TV y disponer de puerto paralelo. Con todo ésto listo, se debe instalar el soporte para el sistema 'Video 4 linux 2', ya que esta versión del kernel aún no lo incorpora, y el driver apropiado para el control de la tarjeta de TV: bttv9.

Para instalar este software debemos visitar la página de su autor y descargar los fuentes comprimidos de las versiones apropiadas (de nuevo, ver el Pliego de Condiciones). Toda vez que estos archivos estén en el PC, sólo habrá que descomprimirlos y compilarlos. Para ello, es preciso seguir las instrucciones que encontraremos acompañando a los fuentes. Tengase en cuenta que se deben instalar como usuario root y que, para su compilación, el kernel debe admitir módulos cargables, tener 'Video 4 linux 1' compilado como módulo y se debe disponer de los ficheros fuente del kernel bajo '/usr/src'.

Para instalar la aplicación en el PC, los únicos pasos que hay que dar consisten en:

1.- Descomprimir el fichero tgz con todos los fuentes, mediante la instrucción:

```
tar -xvzf TFC.tgz
```

2.- Después, cambiar al directorio del proyecto recién creado.

3.- Tomar la identidad de superusuario del sistema (`root`), mediante el comando `su`.

4.- Ejecutar el programa de instalación:

```
./instalaTFC
```

Este *script* ya se encarga de cargar el driver para la comunicación con el Neuron en el kernel del sistema operativo, compilar la aplicación y dejar todos los ficheros necesarios en el directorio 'exes_TFC'.

5.- **IMPORTANTE:** Antes de poder ejecutar el programa se tienen que introducir los sigientes comandos para cargar los drivers del neuron y de la cámara:

```
cd ../neuron2.4
modprobe bttv9
./neuron_load
cd ../exes_TFC
```

Tras ejecutar el script de instalación, en el directorio 'exes_TFC' encontraremos:

- ✓ **navega**: Fichero ejecutable que contiene el programa principal. Será el fichero que se debe ejecutar para arrancar la aplicación, mediante la siguiente sintaxis:

```
navega nodo_destino Fichero_mapa
```

donde:

nodo_destino será un número de 3 cifras válido correspondiente a un nodo que se encuentre en el mapa, seguido de un guión '-' y la letra correspondiente al edificio donde se encuentre el nodo ('N', 'S', 'E' u 'O'), y

Fichero_mapa será el nombre del fichero que contiene la relación de nodos del edificio junto con sus características.

Un ejemplo de llamada a la aplicación que nos guiaría desde donde nos encontrásemos hasta el fondo de pasillo 4 del 3er piso del edificio oeste (nodo 349-O) sería:

```
./navega 349-O EpOeste.map
```

- ✓ **vision**: Ejecutable correspondiente al proceso de visión. Por sí mismo no hace nada, precisa ser arrancado por 'navega'.

- ✓ **odometría**: Ejecutable correspondiente al proceso de odometría. Por sí mismo no hace nada, también precisa ser arrancado por 'navega'.

- ✓ **EpOeste.map**: Fichero con el mapa de nodos del edificio oeste de la Escuela Politécnica de la UAH.

- ✓ **Calib.txt**: Fichero que contiene los parámetros de calibración de la cámara.

III.2. Software de la silla de ruedas (Neuron Chip)

Al igual que en el PC, se debe instalar en los nodos de la red LonWorks el software necesario que se encargue de interpretar nuestras consignas y darnos los datos de velocidad de las ruedas y tiempo acumulado. En el proyecto [Sebastián-99] se presenta paso a paso cual es el método de trabajo con esta herramienta.

En el apartado de planos se encuentra el código que hay que cargar en el nodo Neuron Chip que sirve de interfaz entre la aplicación y la red de control de la silla de ruedas. Una vez compilado este código, basta con descargarlo en los nodos tal y como se indica en [Sebastián-99].

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.

IV.1. Equipos físicos.

- Ordenador PC (para el desarrollo y la ejecución de la aplicación)

Microprocesador	Pentium III o superior
Velocidad de reloj	1000 MHz o mayor
Memoria RAM	256 MB
Disco duro	1’5 GB

- Impresora HP Laserjet 2100

Modo de impresión	Láser
Velocidad de páginas	16ppm
Buffer de entrada	4Mbytes
Comunicaciones	Serie y paralelo
Resolución	600ppp

- Tarjeta TV AverMedia.

La tarjeta está basada en el chipset Conexant bt878.

- Cámara Sony XC-73CE.

La cámara usada es una CCD de 1/3in, equipada con una óptica de l = 4.8mm. Las imágenes se capturan en formato VGA, lo que da una resolución de 640 x 480 pixels, para un ancho efectivo del sensor de 4.7mm.

- Silla de ruedas GARANT del modelo 63E-PRO.

Incluye dos motores de tracción del tipo GP76.50Br0.4, cuya alimentación es de 24V. Además, 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 y una batería de alimentación de 24V para el sistema completo.

- Equipo de desarrollo de redes LonWorks (LonBuilder),

Consta de los siguientes elementos:

  - **Network Manager** (gestor de red), es el gestor central del equipo de desarrollo.
  - **Protocol Analyzer** (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*.

IV.2. Equipos lógicos.

- Sistema operativo Linux Mandrake 10.0 con kernel v2.4.22-10mdk[19].
- 'Video 4 Linux 2' (v 2004-07-23 build 010612)
- Driver bttv 0.9.15
- Compilador gcc 3.3.2
- Depurador gdb 6.0-2mdk
- Depurador gestión de memoria valgrind 2.1.0
- Editores de texto vi, kwrite.
- Procesador de textos OpenOffice.org Writer.
- Compilador y depurador de NeuronC (proporcionado por Echelon)

[19] Los núcleos de la serie 2.4 compilados por Mandrake (mdk) destacan porque, entre otras particularidades, se les ha aplicado un parche de baja latencia y tienen todas las opciones compiladas como módulos para así añadirlas cuando convenga vía sus utilidades de configuración.

V. Planos

V.1. Script de compilación

```
cd ./matrices
make
cd ../vision
gcc -g -Wall -I../matrices -I../comunicaciones -c -o barras.o barras.c
gcc -g -Wall -I../matrices -I../comunicaciones -c -o fcirc.o fcirc.c
gcc -g -Wall -I../matrices -I../comunicaciones -c -o fclst.o fclst.c
gcc -g -Wall -I../matrices -I../comunicaciones -c -o fsubwin.o fsubwin.c
gcc -g -Wall -I../matrices -I../comunicaciones -c -o proceso.o proceso.c
gcc -g -Wall -I../matrices -I../comunicaciones -c -o fctr.o fctr.c
gcc -g -Wall -I../matrices -I../comunicaciones -c -o calibra.o calibra.c
gcc -g -Wall -I../matrices -I../comunicaciones -c -o fbrk.o fbrk.c
gcc -g -Wall -I../matrices -I../comunicaciones -c -o posicion.o posicion.c
cp vision ../exes_TFC
cd ../ekf
gcc -g -Wall -I../control -I../matrices -c -o ekf.o ekf.c
cd ../control/odometria
gcc -g -Wall -I../../comunicaciones -I../../matrices -o odometria odometria.c -lrt
cp odometria ../../exes_TFC
cd ..
gcc -g -Wall -I../comunicaciones -I../ekf -c -o utilnodo.o utilnodo.c
gcc -g -Wall -I../comunicaciones -I../ekf -c -o planruta.o planruta.c
gcc -g -Wall -I../comunicaciones -I../ekf -c -o genera.o genera.c
gcc -g -Wall -I../comunicaciones -I../ekf -c -o controla.o controla.c
gcc -g -Wall -I../comunicaciones -I../ekf -c -o gestor.o gestor.c
gcc -g -Wall -I../comunicaciones -I../ekf -lm -lrt -o navega navega.c *.o ../ekf/ekf.o \
 ../matrices/libmatrices.o

cp navega ../exes_TFC
cd ../exes_TFC
```

V.2. Código de la aplicación

colas.h

```
#define KEY              31416
#define MAX_SEND_SIZE    80

#define MSG_ODOM         1
#define MSG_VISION       2
#define MSG_MAPA         3
```

```
struct buf_msg_vi
{
        long mtype;
        char nodo[6];
        float datos[3];
};

struct buf_msg_od
{
        long mtype;
        float datos[3];
};

struct buf_msg_map
{
        long mtype;
        char edificio;
};
```

comunicaciones.h

```
#define TAM_PAQUETE_PC_N 7
#define TAM_PAQUETE_N_PC 8

#define FICH_NEURON      "/dev/neuron0"

typedef union
{
        char contenido[TAM_PAQUETE_N_PC];
        struct
        {
                unsigned short ta_dcha;
                short va_dcha;
                unsigned short ta_izda;
                short va_izda;
        } n_pc;
} paquete_n_pc;

typedef union
{
        char contenido[TAM_PAQUETE_PC_N];
        struct
        {
                char comando;
                char longitud;
                short   v_dcha;
                short   v_izda;
                char eom;
        } pc_n;
} paquete_pc_n;
```

odometria.c

```
#include <sys/time.h>
#include <unistd.h>

#include <sys/types.h>
#include <sys/stat.h>
#include <sched.h>
#include <fcntl.h>
#include <sys/mman.h>
#include <signal.h>
#include <time.h>
#include <sys/ipc.h>
#include <sys/msg.h>

#include "libmatrices.h"
#include "comunicaciones.h"
#include "colas.h"

#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 KCONV    598.628         // para convertir a mrad/s:
                                 // mrad/s=(pulsos*KCONV)/Ts(en ticks)

signed short Resta_Enteros(signed short Anterior,signed short Siguiente);

void manejador(int signal)
{}      // No hace nada, es sólo para despertar al proceso cuando llegue la señal

void manejador2 (int signal)
{
        int res;
```

```
        res = munlockall ();
        if (res==-1)
        {
              perror ("\nOdometria:Fallo munlockall");
              exit (1);
        }
        else printf ("\nOdometria:Páginas desbloqueadas");

        exit (0);
}

int main()
{

        signed int EncoDerecha,EncoIzquierda;
        unsigned int TicksDerecha,TicksIzquierda;
        static signed short EncoDereAnt=0;                // ¡OJO! Definidas como _static_
        static signed short EncoIzqAnt=0;
        static unsigned short TicksDereAnt=0;
        static unsigned short TicksIzqAnt=0;
        float V,Omega,WDerecha,WIzquierda;
        static float V_ant=0,Omega_ant=0;
        int res;
        signed long Aux1,Aux2;
        static long t_aux, tiempo_anterior=0;
        float dif_tiempos;
        struct timespec tiempo_leido;

        struct sched_param planif;
        pid_t id,idcontrol;
        int colaid;
        struct buf_msg_od paquete;

        int idneuron;
        paquete_n_pc paq_neuron;

        id=getpid();
        idcontrol=getppid();     // ID del proceso padre(control)

        //****************** BLOQUEO MEMORIA ************************************

        res = mlockall (MCL_CURRENT);
        if (res==-1)
        {
                perror ("\nOdometria:Fallo mlockall con MCL_CURRENT");
                exit (1);
        }
        else printf ("\nOdometria:Bloqueadas las páginas actuales");

        res = mlockall (MCL_FUTURE);
        if (res==-1)
        {
                perror ("\nOdometria:Fallo mlockall con MCL_FUTURE");
                res = munlockall ();
                if (res==-1)
                {
                        perror ("\nOdometria:Fallo munlockall");
                        exit (1);
                 }
                else printf ("\nOdometria:Páginas desbloqueadas");
                exit (1);
        }
        else printf ("\nOdometria:Bloqueadas las páginas futuras");

//****************** PLANIFICACIÓN ************************************

        planif.sched_priority=sched_get_priority_max(SCHED_FIFO)-1;
        // Para "RT", Odometria, prioridad maxima-1

        res=sched_setscheduler(id,SCHED_FIFO,&planif);
        if (res==-1)
        {
                perror ("\nOdometria: ERROR al cambiar la política de planificación");
                exit (1);
        }
        else printf ("\nOdometria: Planificación RR\n");

        signal (SIGRTMIN+3, manejador);           // Para despertar cuando se llame al modulo
        signal (SIGRTMIN+2, manejador2);          // Para salir cuando se finalice

        if((colaid = msgget(KEY, IPC_CREAT|0660)) == -1)
        {
                perror("Odometría: No puedo acceder a la cola de mensajes (msgget)");
                exit (1);
        }
```

```
//****************** FIN PLANIFICACIÓN ************************************
        // Abro fichero neuron para escritura
        idneuron=open (FICH_NEURON, O_RDONLY);
        if (idneuron==-1)
        {
                perror ("\nodometria: ERROR - No puedo abrir el NEURON\n");
                return -1;
        }

        // inicialización
        res=clock_gettime(CLOCK_REALTIME, &tiempo_leido);
        tiempo_anterior=tiempo_leido.tv_nsec;

        kill (idcontrol,SIGRTMIN+4); // Devuelvo la CPU al proceso ppal de control

        while (1)
        {
                // recibo un paquete del neuron
                res=read (idneuron, paq_neuron.contenido, TAM_PAQUETE_N_PC);
                if (res==-1)
                        perror ("\nError al leer del NEURON\n");

                EncoDerecha=paq_neuron.n_pc.va_dcha;
                TicksDerecha=paq_neuron.n_pc.ta_dcha;
                EncoIzquierda=paq_neuron.n_pc.va_izda;
                TicksIzquierda=paq_neuron.n_pc.ta_izda;

                // ahora se obtiene la velocidad en mrad/s...
                EncoDerecha=Resta_Enteros(EncoDereAnt,EncoDerecha);
                EncoIzquierda=Resta_Enteros(EncoIzqAnt,EncoIzquierda);
                TicksDerecha=Resta_Enteros(TicksDereAnt,TicksDerecha);
                TicksIzquierda=Resta_Enteros(TicksIzqAnt,TicksIzquierda);

                if (!(TicksDerecha==0||TicksIzquierda==0))
                {

                        if (EncoDereAnt==0 && EncoIzqAnt==0 && TicksDereAnt==0 && TicksIzqAnt==0)
                        {
                                // La primera iteración es de inicializacion
                                printf("\nInicializa odometria");
                                fflush (stdout);

                                EncoDerecha=0;
                                EncoIzquierda=0;
                                TicksDerecha=1;
                                TicksIzquierda=1;
                        }

                        // Almaceno para proxima vez la lectura de encoder
                        EncoDereAnt=paq_neuron.n_pc.va_dcha;
                        TicksDereAnt=paq_neuron.n_pc.ta_dcha;
                        EncoIzqAnt=paq_neuron.n_pc.va_izda;
                        TicksIzqAnt=paq_neuron.n_pc.ta_izda;


                        Aux1=(signed long)EncoDerecha;
                        Aux2=(signed long)TicksDerecha;
                        if(Aux2==0)
                                puts("Error: Ticks derecha son 0");

                        WDerecha=(float)((Aux1*KCONV)/Aux2);
                        Aux1=(signed long)EncoIzquierda;
                        Aux2=(signed long)TicksIzquierda;
                        if(Aux2==0)
                                puts("Error: Ticks izquierda son 0");

                        WIzquierda=(float)((Aux1*KCONV)/Aux2);

                        // Calculo el tiempo transcurrido desde la última medida
                        res=clock_gettime(CLOCK_REALTIME, &tiempo_leido);
                        if (res==-1)
                        {
                                tiempo_leido.tv_nsec=tiempo_anterior+50000000;
                                if (tiempo_leido.tv_nsec<50000000)
                                        tiempo_leido.tv_sec++;
                        // Si hay un error al leer el tiempo, tomamos 50ms que es
                        // el que debe ser
                        }

                        // 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.

                }
                else    // POSIBLE ERROR!!
                {
                        V=V_ant;
```

```
                        Omega=Omega_ant;
                }

                // Tiempo transcurrido desde la ultima medida (pasado a segundos)
                t_aux=(tiempo_leido.tv_nsec>tiempo_anterior) ?
                tiempo_leido.tv_nsec-tiempo_anterior :
                tiempo_leido.tv_nsec-tiempo_anterior+1000000000;

                dif_tiempos=(float)t_aux/1000000000;

                tiempo_anterior=tiempo_leido.tv_nsec;// Lo guardamos para la siguiente iteración

                V_ant=V;
                Omega_ant=Omega;

                // enviar_la_info_necesaria : Se envia V,Omega y dif_tiempos
                paquete.mtype=MSG_ODOM;
                paquete.datos[0]=V;
                paquete.datos[1]=Omega;
                paquete.datos[2]=dif_tiempos;

                res=msgsnd(colaid,(struct msgbuf *)&paquete,
                                sizeof(paquete)-sizeof(long), IPC_NOWAIT);
                if (res==-1)
                        perror ("\nOdometria: Error al enviar mensaje");

                sleep(5);       // 5 segs serán suficientes, nunca se harán enteros
                                // Cuando se reciba señal se vuelve a empezar
        } // del while (1)
        return 0;
}

//---------------------------------------------------------------
// 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 short Resta_Enteros(signed short Anterior,signed short Siguiente)
{
        signed long Incremento;
        signed short  Retorno;

        Incremento=(signed long)(Siguiente-Anterior);
        if(Anterior>Siguiente)
                        Incremento+=65536;

        Retorno=(signed short)Incremento;

        return Retorno;
}
```

controla.h

```
#include <string.h>
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>

#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 KCONV   598.628         // para convertir a mrad/s:
                                // mrad/s=(pulsos*KCONV)/Ts(en ticks)
#define KD      90              // (15*90) Cte error desplazamiento.
                                // Si error 1/1cm escalados=5m-> 90º, 0.18º/cm
#define KO      0.007*90
        // (0.001*90) Cte error orientacion. Si error de 1/0.007º=142 ->90º, 0.63º/º

#define OMEGA_MAX               150             // en grados/s
#define VGIRO_MAX_ACELERA       0.07            // en rad/s   era 0.1

#define KO_DESORIENTO    0.011
// Cte error orientacion si esta perdido. Si es de 90 grados la saldia es maxima

//ctes de controlador V
#define VLINEAL_MAX             0.7     // en m/s
```

```
#define VLINEAL_GIRO             0.1     // en m/s
#define VLINEAL_MAX_ACELERA      0.005   // en m/s



#define MAX_TAREAS               10// Mx nº de tareas que puede tener la trayectoria entre 2 nodos.
#define AVANCE                   1 // Estado indica que la tarea es de avance.
#define GIRO                     2 // Estado que indica que la tarea es de giro.


#define ENTORNO                  0.01    // Medio metro
#define CERCA                    0.1

#define min(a, b)        ((a)<(b)) ? (a) : (b)
#define max(a, b)        ((a)>(b)) ? (a) : (b)

/*****************************************************************/
/*              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 TareaActual[MAX_TAREAS];
extern int BufferActual;
extern int BufferProximo;

extern FILE *Sigueme;
extern FILE *Consigna;
extern FILE *Movil;
```

controla.c

```
#include "controla.h"
#include "comunicaciones.h"

static int NumTarea=0;
static Posicion PosicionDeseo;
extern int idneuron;


//---------------------------------------------------------------
// 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.(rad/s)
//               float ConsignaV  -> Consigna de velocidad lineal a enviar.(m/s)
// Parametros salida:
//               No tiene pues solo actua sobre el puerto paralelo.
//---------------------------------------------------------------

void Envia_Consignas(float ConsignaOmega, float ConsignaV)
{
        int res;

        paquete_pc_n paq;

        paq.pc_n.comando=0x01;
        paq.pc_n.longitud=4;
        paq.pc_n.eom=0x00;

        paq.pc_n.v_dcha=(short)(((ConsignaV+(ConsignaOmega*(DR/2)))/RR)*1000);// Envío mrad/seg
        paq.pc_n.v_izda=(short)(((ConsignaV-(ConsignaOmega*(DR/2)))/RR)*1000);// Envío mrad/seg
```

```
        res=write (idneuron, paq.contenido, TAM_PAQUETE_PC_N);
        if (res==-1)
                perror ("\nError al escribir al NEURON\n");

        return;  // Fin de la función
}

//---------------------------------------------------------------
// 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:
// Parametros salida:
//               Posicion -> estructura con la posicion pedida
//---------------------------------------------------------------

Posicion Posicion_Deseo_Avance(void)
{
        Posicion PosicionDeseoLocal;
        float m1,m2,b1,b2;

        if((TareaActual[NumTarea].Origen.Theta==90) || (TareaActual[NumTarea].Origen.Theta==270))
        {
                PosicionDeseoLocal.x=TareaActual[NumTarea].Origen.x;
                PosicionDeseoLocal.y=PosicionReal.y;
        }
        else if
        ((TareaActual[NumTarea].Origen.Theta==0) || (TareaActual[NumTarea].Origen.Theta==180))
        {
                PosicionDeseoLocal.y=TareaActual[NumTarea].Origen.y;
                PosicionDeseoLocal.x=PosicionReal.x;
        }
        else
        {
                m1=tan(TareaActual[NumTarea].Origen.Theta*PI/180);
                m2=-1/m1;   // la normal

                b1=TareaActual[NumTarea].Origen.y-m1*TareaActual[NumTarea].Origen.x;
                b2=PosicionReal.y-m2*PosicionReal.x;

                PosicionDeseoLocal.x=(b1-b2)/(m2-m1);
                PosicionDeseoLocal.y=m1*PosicionDeseoLocal.x+b1;
        }

        PosicionDeseoLocal.Theta=TareaActual[NumTarea].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:
// 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;

        // recojo parametros de giro
        xCentro=TareaActual[NumTarea].Circulo.xCentro;
        yCentro=TareaActual[NumTarea].Circulo.yCentro;
        R=TareaActual[NumTarea].Circulo.Radio;

        // obtengo el punto a traves de la ec. de la circunferencia y
        // la recta que une el punto real y el origen
        if(PosicionReal.y==yCentro)
        {
                PosicionDeseoLocal.y=PosicionReal.y;
                PosicionDeseoLocal.x=xCentro+R;
        }

        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;
                }
        }

        // despues he de hallar el angulo deseado. Se obtiene facilmente a traves
        // de la pendiente de la recta que 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(TareaActual[NumTarea].Destino.Theta,TareaActual[NumTarea].Origen.Theta);
        MinThetaTramo=min(TareaActual[NumTarea].Destino.Theta,TareaActual[NumTarea].Origen.Theta);

        // amplio el rango para los entornos de los puntos de origen y fin del trayecto
        if((MaxThetaTramo-MinThetaTramo)<180)   // estoy fuera de la zona de discontinuidad
        {
                MaxThetaTramo=min(MaxThetaTramo+2,360);   // el 2 es un entorno de error
                if (MinThetaTramo!=0)
                        MinThetaTramo=max(MinThetaTramo-2,0);
                else
                // si es exactamente 0, pasamos de no estar en zona de discontinuidad a si estarlo
                {
                        MinThetaTramo=MaxThetaTramo;
                        MaxThetaTramo=358;
                }
        }
        else    // estoy dentro de la zona de discontinuidad
        {
                MaxThetaTramo=MaxThetaTramo-2;
                MinThetaTramo=MinThetaTramo+2;
        }

        // Ahora calculo la consigna de orientacion
        // 1º elimino indeterminaciones.....
        if(PosicionReal.x==xCentro)
                PosicionDeseoLocal.Theta=0;
        else if(PosicionReal.y==yCentro)
                        PosicionDeseoLocal.Theta=90;
             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
        if((MaxThetaTramo-MinThetaTramo)<180)
                if
                ((PosicionDeseoLocal.Theta<MinThetaTramo)||
                (PosicionDeseoLocal.Theta>MaxThetaTramo))
                        PosicionDeseoLocal.Theta=fmod(PosicionDeseoLocal.Theta+180,360);
        else // estoy en zona de discontinuidad, la condicion es la opuesta
                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
//              TareaActual de tipo Trayectoria.
//              Lo mismo ocurre con las funciones Posicion_Deseo_Avance/Giro
// Parametros salida:
//              float -> velocidad angular a aplicar.
//-----------------------------------------------------------------

float Controla_Omega(void)
{
        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.
        //----------------------------------------

        ConsignaOmega=ErrorDesplazamiento*KD+KO*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;

        // CAMBIO DE FONDO DE ESCALA, seria lo mismo que V
        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:
//              La info que necesita     se encuentra en el array de estructuras global
//              TareaActual de tipo Trayectoria
// Parametros salida:
//              float -> velocidad lineal a aplicar.
//-----------------------------------------------------------------

float Controla_V(void)
{
        float ConsignaV, DistanciaDestino;

        if(TareaActual[NumTarea].Status==AVANCE)
        {  // Calculo la distancia Euclidea hasta el final
                DistanciaDestino=
                pow((TareaActual[NumTarea].Destino.x-PosicionReal.x),2)+
                pow((TareaActual[NumTarea].Destino.y-PosicionReal.y),2);

                DistanciaDestino=sqrt(DistanciaDestino);

                // CAMBIO DE FONDO DE ESCALA, LO MAXIMO ES VLINEAL_MAX
                // solo depende de la distancia cuando estoy a 5m de destino
                if(DistanciaDestino<=0.5)
```

```
                        if(TareaActual[NumTarea+1].Status!=-1)
                                ConsignaV=VLINEAL_GIRO*TareaActual[NumTarea+1].Circulo.Radio*3;
                else
                        ConsignaV=VLINEAL_GIRO;
                        //DistanciaDestino*(VLINEAL_MAX-VLINEAL_MIN)+VLINEAL_MIN; (en m/s)
                }
                else ConsignaV=VLINEAL_MAX;

        }

        // si es tarea de giro va solo a 0.1m/s
        else if(TareaActual[NumTarea].Status==GIRO)
                ConsignaV=VLINEAL_GIRO;

        // 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
//              TareaActual 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:
//              Se pasan las consignas de velocidad por referencia
//              No habra paramertos de salida
// Parametros salida:
//-----------------------------------------------------------------

void Filtro_Aceleracion(float *VeloGiro, float *VeloAvance)
{
        static float VeloAvanceAnterior=0;
        static float VeloGiroAnterior=0;

        // con velocidad de avance
        if((*VeloAvance-VeloAvanceAnterior)>VLINEAL_MAX_ACELERA)
                *VeloAvance=VeloAvanceAnterior+VLINEAL_MAX_ACELERA;

        else if((*VeloAvance-VeloAvanceAnterior)<((-1)*VLINEAL_MAX_ACELERA))
                *VeloAvance=VeloAvanceAnterior-VLINEAL_MAX_ACELERA;

         VeloAvanceAnterior=*VeloAvance;

        // con velocidad de  giro
        if((*VeloGiro-VeloGiroAnterior)>VGIRO_MAX_ACELERA)
                *VeloGiro=VeloGiroAnterior+VGIRO_MAX_ACELERA;

        else if((*VeloGiro-VeloGiroAnterior)<((-1)*VGIRO_MAX_ACELERA))
                *VeloGiro=VeloGiroAnterior-VGIRO_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 TareaActual de tipo Trayectoria
// Parametros salida:
//              Tampoco tiene.
//-----------------------------------------------------------------

void Controla(void)
{
        static float ConsignaOmega=0;
        static float ConsignaV=0;
        float MaxTheta, MinTheta,Distancia;
        char NumeroFP[20];

        static int Estoy_cerca=0;

        //----------------------------------------
        // 1º se comprueba que no he acabado toda la trayectoria
        //----------------------------------------

        if(BufferActual==1)
        {
                //----------------------------------------
                // 2º se obtiene la posicion deseada (consigna en este instante)
                //----------------------------------------

                if(TareaActual[NumTarea].Status==AVANCE)
                        PosicionDeseo=Posicion_Deseo_Avance();
                else if(TareaActual[NumTarea].Status==GIRO)
                        PosicionDeseo=Posicion_Deseo_Giro();

                //----------------------------------------
                // 3º 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);

                //----------------------------------------
                // 4º 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))
```

```
                {
                        printf ("\nMovil desorientado 1");
                        ConsignaOmega=Movil_Desorientado();
                        ConsignaV=0;     // y lo paro
                }

                // sino es que esta dentro.
                else
                {       ConsignaOmega=Controla_Omega();
                        ConsignaV=Controla_V();
                }
        }

        // el entorno no incluye la zona de discontinuidad.
        else
        {       // esta fuera del rango +-90º
                if((PosicionReal.Theta<MinTheta)||(PosicionReal.Theta>MaxTheta))
                {
                        printf ("\nMovil desorientado 2");
                        ConsignaOmega=Movil_Desorientado();
                        ConsignaV=0;     // y lo paro
                }

                // sino esta dentro.
                else
                {       ConsignaOmega=Controla_Omega();
                        ConsignaV=Controla_V();
                }
        }

        //----------------------------------------
        // 5º se comprueba que no se ha alcanzado el destino de la tarea actual
        //----------------------------------------
        Distancia=pow((TareaActual[NumTarea].Destino.x-PosicionReal.x),2)+
                  pow((TareaActual[NumTarea].Destino.y-PosicionReal.y),2);

        printf ("\nDistancia: %f",Distancia);
        fflush(stdout);

        if (Estoy_cerca && (Distancia>CERCA) )
        {
                // Me pasé
                NumTarea++;
                printf( "\nMe pasé.Doy la tarea nº %i por completada\n",NumTarea-1);
                if(TareaActual[NumTarea].Status==-1)
                {
                        BufferActual=0;
                        NumTarea=0;
                        puts("Afrontando nuevo tramo");
                }
        }
        else if (Distancia < CERCA)
                 Estoy_cerca=1;

        if  (Distancia<=ENTORNO) //ALREDEDOR_X&&ALREDEDOR_THETA&&ALREDEDOR_Y)
        {
                NumTarea++;
                Estoy_cerca--;

                printf( "\n tarea nº %i completada\n",NumTarea-1);

                // Ademas se comprueba si se ha completado un tramo entre dos nodos
                if(TareaActual[NumTarea].Status==-1)
                {       BufferActual=0;
                        NumTarea=0;
                        puts("Afrontando nuevo tramo");

                }
        }


}

// Si no hay tarea entonces se para el movil
else
{
        ConsignaOmega=0;
        ConsignaV=0;
}


Filtro_Aceleracion(&ConsignaOmega,&ConsignaV);

if((BufferActual==0 && BufferProximo==0))
{
        ConsignaV=0;
        ConsignaOmega=0;
```

```
                puts("Se acabo");
        }
        Envia_Consignas(ConsignaOmega,ConsignaV);

        //----------------------------------------
        // 6º escribo vbles en sigueme.m y Consigna.m
        //----------------------------------------
/* Si quiero ver la posicion deseada implementada por el generador*/
        gcvt(PosicionDeseo.x, 8, NumeroFP);
        fprintf(Sigueme,"%s ",NumeroFP);
        gcvt(PosicionDeseo.y, 8, NumeroFP);
        fprintf(Sigueme,"%s ",NumeroFP);
        gcvt(PosicionDeseo.Theta, 8, NumeroFP);
        fprintf(Sigueme,"%s \n",NumeroFP);

        gcvt(ConsignaOmega, 8, NumeroFP);
        fprintf(Consigna,"%s ",NumeroFP);
        gcvt(ConsignaV, 8, NumeroFP);
        fprintf(Consigna,"%s \n",NumeroFP);

        /* Fin de la funcion Controla */
}
```

genera.h

```
#include <string.h>
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define PI               3.1415926
#define MAX_LONG_NODO    200
#define MAX_TAREAS       10

#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))

#define AVANCE   1
#define GIRO     2

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 TareaProxima[MAX_TAREAS];      // las que va preparando el generador

void Buscar_Nodo(int NombreNodo, char* result);
```

genera.c

```
#include "genera.h"

static float ConsignaGiro[3];
```

```
static float ConsignaAvance[3];

//-------------------------------------------------------------
// Nombre: Tarea_Giro
// Funcion:
//      Tarea que programa el controlador para avanzar en linea recta
//              hasta el destino.
// Parametros entrada:
//              float 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
/*-------------------------------------------------------------*/

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;
        }
        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
        //---------------------------------------------
        if(yNodo2==yNodo1) //Elimino indet. de Ay=0
        {
                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
        }

        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;
        }
        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;
        }

        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:
//              int NumTarea -> el numero de la tarea para la que genera los parametros
// Parametros salida:
//              Ninguno, es global el array de tareas.
/*---------------------------------------------------------------*/

void Parametros_Giro(int NumTarea)
{
        float m1,m2,b1,b2;
        float xCentro, yCentro,R;

        //-----------------------------------------
        // Despues obtengo el origen de la circunferencia
        //-----------------------------------------
        // elimino indeterminaciones
        if      ((TareaProxima[NumTarea].Origen.Theta!=0) &&
                (TareaProxima[NumTarea].Origen.Theta!=180) &&
                (TareaProxima[NumTarea].Destino.Theta!=0) &&
                (TareaProxima[NumTarea].Destino.Theta!=180))
        {
                if      ((TareaProxima[NumTarea].Origen.Theta==90) ||
                        (TareaProxima[NumTarea].Origen.Theta==270))
                        m1=0;
                else
                {
                        m1=tan(TareaProxima[NumTarea].Origen.Theta*PI/180);
                        m1=-1/m1;   // la normal
                }

                if      ((TareaProxima[NumTarea].Destino.Theta==90) ||
                        (TareaProxima[NumTarea].Destino.Theta==270))
                        m2=0;
                else
                {
                        m2=tan(TareaProxima[NumTarea].Destino.Theta*PI/180);
                        m2=-1/m2;   // la normal
                }

                b1=TareaProxima[NumTarea].Origen.y-m1*TareaProxima[NumTarea].Origen.x;
                b2=TareaProxima[NumTarea].Destino.y-m2*TareaProxima[NumTarea].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 division por 0
        }

        // la pendiente de la normal al punto de origen es infinito
        else if((TareaProxima[NumTarea].Destino.Theta!=0)&&
                (TareaProxima[NumTarea].Destino.Theta!=180))
        {
                xCentro=TareaProxima[NumTarea].Origen.x;

                if((TareaProxima[NumTarea].Destino.Theta==90) ||
                  (TareaProxima[NumTarea].Destino.Theta==270))
                        m2=0;
                else
                {
                        m2=tan(TareaProxima[NumTarea].Destino.Theta*PI/180);
                        m2=-1/m2;   // la normal
                }

                b2=TareaProxima[NumTarea].Destino.y-m2*TareaProxima[NumTarea].Destino.x;
                yCentro=m2*xCentro+b2;
        }
        // la pendiente de la normal al punto de destino es infinito
        else if((TareaProxima[NumTarea].Origen.Theta!=0) &&
                (TareaProxima[NumTarea].Origen.Theta!=180))
        {
                xCentro=TareaProxima[NumTarea].Destino.x;

                if      ((TareaProxima[NumTarea].Origen.Theta==90) ||
                        (TareaProxima[NumTarea].Origen.Theta==270))
```

```
                        m1=0;
                else
                {
                        m1=tan(TareaProxima[NumTarea].Origen.Theta*PI/180);
                        m1=-1/m1;   // la normal
                }
                b1=TareaProxima[NumTarea].Origen.y-m1*TareaProxima[NumTarea].Origen.x;
                yCentro=m1*xCentro+b1;
        }

        //-----------------------------------------
        // Finalmente hallo el radio
        //-----------------------------------------
        R=pow((TareaProxima[NumTarea].Origen.x-xCentro),2)+
          pow((TareaProxima[NumTarea].Origen.y-yCentro),2);
        R=sqrt(R);

        // añadiendo parametros de giro
        TareaProxima[NumTarea].Circulo.xCentro=xCentro;
        TareaProxima[NumTarea].Circulo.yCentro=yCentro;
        TareaProxima[NumTarea].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)
{
        char NodoOrigen[MAX_LONG_NODO];
        char NodoDestino[MAX_LONG_NODO];
        int NumTarea, HayGiro, AvancePrimera, i;
        int Origen1,Destino1,Origen2, Origen3, Destino2, Destino3;
        float xDestino, yDestino, xOrigen, yOrigen, OrientaOrigen, OrientaDestino;
        float DistanciaNodoDestino, DistanciaNodoOrigen, OrientaDestinoFin;
        float xP1, yP1, xP2, yP2;
        float Parametros[6];
        static Posicion  FinTrayectoAnterior;
        double Aux1,Aux2;

        printf ("\nGenerando trayecto entre %d y %d",Origen,Destino);

        // reseteo el numero de tareas, y los flags
        NumTarea=0;
        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.
        Buscar_Nodo(Origen,NodoOrigen);

        // Si no es un nodo fantasma/transicion
        if(Origen>=0)
        {       // recojo info asociada al punto de acceso
                strtok (NodoOrigen," ");// dummy
                xOrigen=((float)atoi(strtok('\0'," ")))/100;//Obtengo el valor del x del nodo origen
                yOrigen=((float)atoi(strtok('\0'," ")))/100;//Obtengo el valor del y del nodo origen

                OrientaOrigen=(float)atoi(strtok('\0'," "));
                        //Obtengo la orientacion de la normal en grados
                DistanciaNodoOrigen=((float)atoi(strtok('\0'," ")))/100;
                        //Distancia de aprox al nodo origen
```

```
                // 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
                // Obteniendo el punto de acceso al nodo
                strtok (NodoOrigen," ");// dummy
                xP1=((float)atoi(strtok('\0'," ")))/100;
                        //Obtengo el valor x del nodo origen (fantasma)
                yP1=((float)atoi(strtok('\0'," ")))/100;
                        //Obtengo el valor y del nodo origen (fantasma)

                // la distancia al nodo queda igual, pues se hizo eso en origen
                DistanciaNodoOrigen=((float)atoi(strtok('\0'," ")))/100;
                        //Distancia de aprox al nodo origen
        }

        // 2º recojo informacion sobre el nodo de destino del mapa.
        Buscar_Nodo(Destino,NodoDestino);
        strtok (NodoDestino," ");// dummy
        xDestino=((float)atoi(strtok('\0'," ")))/100;      //Obtengo el valor del x del nodo destino
        yDestino=((float)atoi(strtok('\0'," ")))/100;      //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=(float)atoi(strtok('\0'," "));
                        //Obtengo la orientacion de la normal en grados
                DistanciaNodoDestino=((float)atoi(strtok('\0'," ")))/100;
                        //Distancia de aprox al nodo origen

        //OrientaDestino=fmod(OrientaDestino+180,360); // siempre hay que sumar 180 excepto....
                        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
                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;
                }
                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=((float)atoi(strtok('\0'," ")))/100;
                //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.

        //      1ºaccedo 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.
        TareaProxima[NumTarea].Origen.x=FinTrayectoAnterior.x;
        TareaProxima[NumTarea].Origen.y=FinTrayectoAnterior.y;
        TareaProxima[NumTarea].Origen.Theta=FinTrayectoAnterior.Theta;
        TareaProxima[NumTarea].Destino.x=ConsignaAvance[1];
        TareaProxima[NumTarea].Destino.y=ConsignaAvance[2];
        TareaProxima[NumTarea].Destino.Theta=ConsignaAvance[0];
                // HA DE COINCIDIR CON LA DE ORIGEN
        TareaProxima[NumTarea].Status=AVANCE;
        NumTarea++;

        // Espero a que llegue al nuevo piso.
puts("Silla en ascensor, esperando alcanzar nuevo piso. Pulse <enter>cuando ocurra");
        getchar();

        // 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.
        TareaProxima[NumTarea].Origen.x=TareaProxima[NumTarea-1].Destino.x;
        TareaProxima[NumTarea].Origen.y=TareaProxima[NumTarea-1].Destino.y;
        TareaProxima[NumTarea].Origen.Theta=TareaProxima[NumTarea-1].Destino.Theta;
        TareaProxima[NumTarea].Destino.x=ConsignaAvance[1];
        TareaProxima[NumTarea].Destino.y=ConsignaAvance[2];
        TareaProxima[NumTarea].Destino.Theta=ConsignaAvance[0];
        TareaProxima[NumTarea].Status=AVANCE;

        // y retorno....
        for(i=NumTarea+1;i<MAX_TAREAS;i++)        TareaProxima[i].Status=-1;
        FinTrayectoAnterior=(Posicion)TareaProxima[NumTarea].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.
                Aux1=((double)((long int)(ConsignaGiro[0]*100)))/100;
                Aux2=((double)((long int)(OrientaOrigen*100)))/100;
                if(Aux1!=Aux2)
                {
                        // y actualizando el area para el controlador.
                        TareaProxima[NumTarea].Origen.x=xOrigen;
                        TareaProxima[NumTarea].Origen.y=yOrigen;
                        TareaProxima[NumTarea].Origen.Theta=OrientaOrigen;
                        TareaProxima[NumTarea].Destino.x=ConsignaGiro[1];
                        TareaProxima[NumTarea].Destino.y=ConsignaGiro[2];
                        TareaProxima[NumTarea].Destino.Theta=ConsignaGiro[0];
                        TareaProxima[NumTarea].Status=GIRO;

                        // añadiendo parametros de giro
                        Parametros_Giro(NumTarea);
                        NumTarea++;
                }
                else AvancePrimera=1;

                // 2º Tarea de avance
                // El movil ha de terminar en la misma posicion que necesita como entrada
                // la tarea de giro

                ConsignaAvance[0]=ConsignaGiro[0];

                // calculo la orientacion del nodo destino.....
                // Añadido del nuevo generador.....
                // SOLO SI NO ES NODO DE TRANSICION
                if(Destino>0)    OrientaDestinoFin=fmod(OrientaDestino+180,360); // siempre hay que
sumar 180 excepto....
                else OrientaDestinoFin=OrientaDestino;

                Aux1=((double)((long int)(ConsignaAvance[0]*100)))/100;
                Aux2=((double)((long int)(OrientaDestinoFin*100)))/100;
                // 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
                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
                {


                        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)
                {
                        TareaProxima[NumTarea].Origen.x=xOrigen;
                        TareaProxima[NumTarea].Origen.y=yOrigen;
                        TareaProxima[NumTarea].Origen.Theta=OrientaOrigen;
                        AvancePrimera=0;
                        }
                else
                {       TareaProxima[NumTarea].Origen.x=TareaProxima[NumTarea-1].Destino.x;
                        TareaProxima[NumTarea].Origen.y=TareaProxima[NumTarea-1].Destino.y;
                        TareaProxima[NumTarea].Origen.Theta=TareaProxima[NumTarea-1].Destino.Theta;
                }
                TareaProxima[NumTarea].Destino.x=ConsignaAvance[1];
                TareaProxima[NumTarea].Destino.y=ConsignaAvance[2];
                TareaProxima[NumTarea].Destino.Theta=ConsignaAvance[0];
                TareaProxima[NumTarea].Status=AVANCE;
                NumTarea++;

                // 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)
                {
                        ConsignaGiro[2]=yDestino;
                        ConsignaGiro[1]=xDestino;
                        ConsignaGiro[0]=OrientaDestinoFin;
                        HayGiro=0;

                        // y actualizando el area para el controlador.
                        TareaProxima[NumTarea].Origen.x=TareaProxima[NumTarea-1].Destino.x;
                        TareaProxima[NumTarea].Origen.y=TareaProxima[NumTarea-1].Destino.y;
                        TareaProxima[NumTarea].Origen.Theta=TareaProxima[NumTarea-1].Destino.Theta;
                        TareaProxima[NumTarea].Destino.x=ConsignaGiro[1];
                        TareaProxima[NumTarea].Destino.y=ConsignaGiro[2];
                        TareaProxima[NumTarea].Destino.Theta=ConsignaGiro[0];
                        TareaProxima[NumTarea].Status=GIRO;

                        // añadiendo parametros de giro
                        Parametros_Giro(NumTarea);
                        NumTarea++;
                }


        // y retornamos. Vaciando el array de tareas y guardando el ultimo movimento para ascensores
                for(i=NumTarea;i<MAX_TAREAS;i++)  TareaProxima[i].Status=-1;
                FinTrayectoAnterior=(Posicion)TareaProxima[NumTarea-1].Destino;
                return;
        }

}
```

gestor.h

```
#include <string.h>
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define MAX_NODOS_MAPA                  200
#define MAX_LONG_NODO                   200
#define MAX_LONG_NOMBRE                 3
#define MAX_NODOS_RUTA                  50
#define MAX_TAREAS                      10
#define MAX_X                           MAX_TAREAS*MAX_NODOS_RUTA*2*10
#define TS                              50

extern FILE *Mapa;
extern int Ruta[MAX_NODOS_MAPA];

typedef struct
{       float x;
        float y;
        float Theta;
}Posicion;
extern Posicion PosicionReal;

void Plan_Ruta(char *Origen, char *Destino);
void Genera_Trayecto(int Origen, int Destino);
void init_ekf ();
void ekf (float v, float omega, float tiempo, Posicion *Xanterior, int hay_vision, Posicion
Zanterior);
void fin_ekf ();
void Controla(void);
```

gestor.c

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <time.h>
#include <signal.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include <math.h>
#include "gestor.h"
#include "comunicaciones.h"
#include "colas.h"

#define VECES_VISION     50
#define ESCALA           5       // Para obtener x,y en la escala del mapa. (cada cm son 5m)
```

```
#define PI              3.1415926

int BufferActual; // estado del buffer de tareas actuales. Para controlador
int BufferProximo; // estado del buffer de tareas proximas. Para generador
float ConsignaOmegaBase; // es global poruqe ha de saberla el controlador

typedef struct
{
        float xCentro;
        float yCentro;
        float Radio;
}Giro;

typedef struct
{
        Posicion Origen;
        Posicion Destino;
        Giro Circulo;
        int Status;
}Trayectoria;

Trayectoria TareaActual[MAX_TAREAS];      // las que va describiendo el controller
Trayectoria TareaProxima[MAX_TAREAS];     // las que va preparando el generador

FILE *Proceso;
int idneuron;                   // Se abre aqui, se usa en controla.c y se cierra en navega.c(en el
main)
extern int colaid;              // Viene de navega.c
extern char* f_mapa;

void CorrigeTProcesoVision(Posicion *Pos, float *V,float *omega,float *tiempo,int *veces_vision);

//-----------------------------------------------------------------
// Nombre: Gestor_Trayecto
// Funcion:
//      Realiza la gestion de trayectorias del sistema. Llama al
//              generador de trayectorias y controla el punto en que se
//              encuentra el movil.
//              El es proceso que apodera de la CPU mientras esta andando la silla
// Parametros entrada:
//              Recibe el nodo destino para llamar al planificador de rutas
// Parametros salida:
//              Ninguno, no son necesarios (en ppio)
//-----------------------------------------------------------------

void Gestor_Trayecto(char *Destino)
{
        pid_t idvision, idodom;
        int NumRuta;
        long int Posx, Posy, PosTheta, i;
        char NumeroFP[20];
        char Origen[5];         // Nombre de nodo son 4 caracteres maximo
        FILE *Trayectoria;

        int veces_vision;       // Almacena cuantos veces se ejecuto odometria
                                // desde que se inició la iteración de visión
        float V[VECES_VISION];  // Velocidad lineal
        float omega[VECES_VISION];// Velocidad angular
        float tiempo[VECES_VISION];// Tiempo transcurrido entre la última y la penultima odometria
        Posicion PosicionVision; // Datos de posicion obtenidos via vision
        int res,hay_vision=0;   // Indica si se dispone de datos válidos de vision (1) o no (0)

        // Paquetes cola de mensajes
        struct buf_msg_vi dvision;
        struct buf_msg_od dodom;
        struct buf_msg_map paq_mapa;

        // Temporizador
        struct sigevent evento;
        timer_t idtemp;
        struct itimerspec itspec;

        char *arg[3];
                arg[0]="vision";
                arg[1]=f_mapa;
                arg[2]=(char*)0;

        // Creo proceso hijo (Vision) y obtengo nodo y posición de partida
        paq_mapa.mtype=MSG_MAPA; // Paquete mapa
        paq_mapa.edificio='O';          //Edificio origen
        res=msgsnd(colaid,(struct msgbuf *)&paq_mapa, sizeof(char), IPC_NOWAIT);
        if (res==-1)
                perror ("\nControl: Error al enviar mensaje(mapa)");


        idvision =fork();
        if (idvision==-1)
```

```
{
        perror ("\nControl: Error en el fork\n");
        exit (1);
}

if (idvision==0)
{
        res=execv ("./vision", arg);
        if (res==-1)
        {
                perror ("\nError en el exec\n");
                exit (1);
        }
}
else
{               // Continua el código de control

        do
                res=msgrcv (colaid, &dvision, MAX_SEND_SIZE, MSG_VISION, 0);
                        // Espero a recibir (sin NOWAIT)
        while (!strcmp(dvision.nodo,"---"));

        strcpy(Origen, dvision.nodo);

        Plan_Ruta (Origen, Destino);

        // este fichero va a almacenar todos los mensajes producidos durante el proceso
        if(!(Proceso=fopen("Proceso.txt","w+")))
        {
                puts("Error Abriendo el fichero de procesamiento");
                exit(-1);
        }

        // Me creo un fichero para luego ver resultado en Matlab
        if(!(Trayectoria=fopen("Trayecto.m","w")))
        {
                puts("Error Abriendo el fichero Matlab");
                exit(-1);
        }
        // preparo el fichero de Matlab.
        fseek(Trayectoria,0,SEEK_SET);
        fputs("x=[",Trayectoria);
        Posx=ftell(Trayectoria);

        fseek(Trayectoria,MAX_X,SEEK_SET);
        fputs("\ny=[",Trayectoria);
        Posy=ftell(Trayectoria);

        fseek(Trayectoria,MAX_X*2,SEEK_SET);
        fputs("\nTheta=[",Trayectoria);
        PosTheta=ftell(Trayectoria);

        // Inicialmente los dos buffers estan libres, el de proxima tareas y el de actuales.
        BufferProximo=0; // Permite al generador realizar una nueva planificacion.
        BufferActual=0;  // indica que el controlador esta en espera, no tiene trabajo

        // Para que encuentre el final de la trayectoria.
        for(i=0;i<MAX_TAREAS;i++)
        {
                TareaActual[i].Status=-1;
                // para que el controlador sepa cuando ha acabado su consigna
                TareaProxima[i].Status=-1;
                // status=-1 indica que no se ha generado esta tarea
        }               // asi se sabe cuando no hay mas tareas generadas para este tramo

        // Por otro lado se inicia el trabajo del gestor: controller y generador.
        NumRuta=0;

        // La 1º vez se ejecuta siempre el generador
        Genera_Trayecto(Ruta[NumRuta],Ruta[NumRuta+1]);           //Generando un tramo
        NumRuta++;

        /*Ya no va a ser la posicion del nodo de partida en principio,
          sabemos exactamente donde estamos*/
        PosicionReal.x=dvision.datos[0];
        PosicionReal.y=dvision.datos[1];
        PosicionReal.Theta=dvision.datos[2];

        // escribo en el fichero este primer tramo
        i=0;
        fseek(Trayectoria,Posx,SEEK_SET);
        while(TareaProxima[i].Status!=-1)
        {
                gcvt(TareaProxima[i].Origen.x, 8, NumeroFP);
                fprintf(Trayectoria,"%s ",NumeroFP);
                gcvt(TareaProxima[i].Destino.x, 8, NumeroFP);
                fprintf(Trayectoria,"%s ",NumeroFP);
```

```
                i++;
        }
        Posx=ftell(Trayectoria);

        i=0;
        fseek(Trayectoria,Posy,SEEK_SET);
        while(TareaProxima[i].Status!=-1)
        {
                gcvt(TareaProxima[i].Origen.y, 8, NumeroFP);
                fprintf(Trayectoria,"%s ",NumeroFP);
                gcvt(TareaProxima[i].Destino.y, 8, NumeroFP);
                fprintf(Trayectoria,"%s ",NumeroFP);
                i++;
        }
        Posy=ftell(Trayectoria);

        i=0;
        fseek(Trayectoria,PosTheta,SEEK_SET);
        while(TareaProxima[i].Status!=-1)
        {
                gcvt(TareaProxima[i].Origen.Theta, 8, NumeroFP);
                fprintf(Trayectoria,"%s ",NumeroFP);
                gcvt(TareaProxima[i].Destino.Theta, 8, NumeroFP);
                fprintf(Trayectoria,"%s ",NumeroFP);
                i++;
        }
        PosTheta=ftell(Trayectoria);


        // y se indica al controlador que ya hay consigna que alcanzar (y se copia)
        // a la vez hay que obtener la direccion base de omega
        memcpy(TareaActual,TareaProxima,sizeof(TareaProxima));
        BufferActual=1;          // arranco el controlador

        init_ekf ();
        hay_vision=1;
        PosicionVision.x=dvision.datos[0];
        PosicionVision.y=dvision.datos[1];
        PosicionVision.Theta=dvision.datos[2];
        veces_vision=0;

// Creo proceso hijo: Odometria
        arg[0]="odometria";
        arg[1]=(char*)0;

        idodom =fork();
        if (idodom==-1)
        {
                perror ("\nControl: Error en el fork(2)\n");
                exit (1);
        }

        if (idodom==0)
        {
                res=execv ("./odometria", arg);
                if (res==-1)
                {
                        perror ("\nError en el exec(2)\n");
                        exit (1);
                }
        }
        else
        {               // Continua el código de control

        //Espero señal de que se ha activado (espera inactiva con sleep)
                sleep(5);        // No se llega a dormir los 5 segs

                // Abro el neuron
                idneuron=open (FICH_NEURON, O_WRONLY);
                if (idneuron==-1)
                {
                        perror ("\nERROR: No puedo abrir el NEURON\n");
                        exit (1);
                }
        // Configuro un temporizador para despertar el proceso cada 50 mseg

                evento.sigev_signo=SIGRTMIN+4;
                evento.sigev_notify=SIGEV_SIGNAL;

                res=timer_create(CLOCK_REALTIME, &evento, &idtemp);

                itspec.it_value.tv_sec=0;     // activacion en 50 mseg
                itspec.it_value.tv_nsec=50000000;  // activacion en 50 mseg

                itspec.it_interval.tv_sec=0;       // intervalo de 50 mseg
                itspec.it_interval.tv_nsec=50000000;// intervalo de 50 mseg
```

```
                res=timer_settime (idtemp,0, &itspec,NULL);

                        // OJO! Pruebas (Quitar)
                //kill (idvision,SIGRTMIN); //Finalizo vision


                do
// AQUI EMPIEZA EL BUCLE INFINITO QUE LLAMA CONTINUAMENTE AL CONTROLADOR
                {
                // 1º Enviamos consignas a las ruedas para que la silla se mueva

                        Controla ();

                // 2º Llamamos a Odometría para obtener la info de las ruedas

                        kill (idodom,SIGRTMIN+3);  // SEÑAL A ODOMETRÍA

                        clock_gettime(CLOCK_REALTIME, &tiempoV);

                        // RECIBIR_DATOS_ODOMETRIA() -> V (Queda esperando aquí)
                        res=msgrcv (colaid, &dodom, MAX_SEND_SIZE, MSG_ODOM, 0);

                if (dodom.datos[0]==0 && dodom.datos[1]==0 && dodom.datos[2]==0)
                {
                        // Posible error
                        sleep (1);
                        continue;
                }
                else
                {
                        if (hay_vision==-1)      // iteracion erronea de vision
                                veces_vision=0;

                        V[veces_vision]=dodom.datos[0];
                        omega[veces_vision]=dodom.datos[1];
                        tiempo[veces_vision]=dodom.datos[2];

                // 2.5º Corregir (si la hay) la medida de vision
                        // Se reseteara veces_vision
                        // Se dejará el último valor en la posicion 0 de los arrays
                        if (hay_vision==1)       // iteracion correcta de vision
                                CorrigeTProcesoVision(&PosicionVision,
                                                        V,omega,tiempo,&veces_vision);

                // 3º Llamamos al EKF con los datos que haya disponibles

                        ekf (V[veces_vision], omega[veces_vision], tiempo[veces_vision],
                                 &PosicionReal, hay_vision, PosicionVision);

                // Despues ya es un bucle, hasta que se cumpla todo el recorrido.
                // puedo preparar la proxima tarea, controlador ya esta con la actual
        // Esto gastará tiempo en el primer ciclo de planificacion: control-odom-ekf-vision
        // En el resto, hasta que la tarea de vision acabe, no se hará
                        if((BufferProximo==0) && (Ruta[NumRuta+1]!=0))
                        {
                                Genera_Trayecto(Ruta[NumRuta],Ruta[NumRuta+1]);
                                //Generando un tramo
                                BufferProximo=1;
                                // Linea a quitar cuando no quiero que entre el controlador
                                NumRuta++;
                // al bajar el flag el generador no vuelve a trabajar, hasta que
                // el controlador no ha alcanzado la tarea anterior

                // escribo en el fichero Matlab las x
                                i=0;
                                fseek(Trayectoria,Posx,SEEK_SET);
                                while(TareaProxima[i].Status!=-1)
                                {
                                        gcvt(TareaProxima[i].Origen.x, 8, NumeroFP);
                                        fprintf(Trayectoria,"%s ",NumeroFP);
                                        gcvt(TareaProxima[i].Destino.x, 8, NumeroFP);
                                        fprintf(Trayectoria,"%s ",NumeroFP);
                                        i++;
                                }
                                Posx=ftell(Trayectoria);

                                // escribo en el fichero de MAtlab las y
                                i=0;
                                fseek(Trayectoria,Posy,SEEK_SET);
                                while(TareaProxima[i].Status!=-1)
                                {
                                        gcvt(TareaProxima[i].Origen.y, 8, NumeroFP);
                                        fprintf(Trayectoria,"%s ",NumeroFP);
                                        gcvt(TareaProxima[i].Destino.y, 8, NumeroFP);
                                        fprintf(Trayectoria,"%s ",NumeroFP);
                                        i++;
                                }
```

```
                                Posy=ftell(Trayectoria);

                                i=0;
                                fseek(Trayectoria,PosTheta,SEEK_SET);
                                while(TareaProxima[i].Status!=-1)
                                {
                                        gcvt(TareaProxima[i].Origen.Theta, 8, NumeroFP);
                                        fprintf(Trayectoria,"%s ",NumeroFP);
                                        gcvt(TareaProxima[i].Destino.Theta, 8, NumeroFP);
                                        fprintf(Trayectoria,"%s ",NumeroFP);
                                        i++;
                                }
                                PosTheta=ftell(Trayectoria);
                        }

// Lanzo una tarea del generador cuando este listo el controlador (el pone a 1 el flag)
                        if((BufferActual==0) && (BufferProximo==1))
                        {
                                memcpy(TareaActual, TareaProxima, sizeof(TareaProxima));
                                // se copia nueva tarea
                                BufferActual=1;
                                // se indica al controlador que ya tiene trabajo
                                BufferProximo=0;
                                // se indica al generador que puede crear una nueva tarea
                        }
                        // Aquí se acabarían las "tareas del controlador

                        // Enviar señal a vision si hace falta comenzar iteracion
                        if (hay_vision)
                        {
                                hay_vision=0;
                                kill (idvision, SIGRTMIN+1);
                        }

                // DORMIR HASTA SEÑAL PERIÓDICA 50mseg (se ejecuta vision)

                        sleep(5);

                        // leo a ver si hay vision
                        res=msgrcv (colaid, &dvision, MAX_SEND_SIZE,
                                        MSG_VISION, IPC_NOWAIT);

                        if (res!=-1)
                        {
                                if (!strcmp(dvision.nodo,"---"))
                                {
                                        hay_vision=-1;
                                }
                                else
                                {
                                        hay_vision=1;
                                        PosicionVision.x=dvision.datos[0];
                                        PosicionVision.y=dvision.datos[1];
                                        PosicionVision.Theta=dvision.datos[2];
                                }
                        }

                        veces_vision++;// incrementa el contador

                        }// del else 'no error'
                } while(!(BufferActual==0 && BufferProximo==0));
// FIN DEL BUCLE INFINITO.. CUANDO BufferActual=0 y BufferActual=0
// BufferProximo=0 -> no hay mas tareas....
// BufferActual=0 -> el controlador ya ha acabado

                // acabando con el fichero de Matlab
                fseek(Trayectoria,Posx,SEEK_SET);
                fputs("];",Trayectoria);
                fseek(Trayectoria,Posy,SEEK_SET);
                fputs("];",Trayectoria);
                fseek(Trayectoria,PosTheta,SEEK_SET);
                fputs("];",Trayectoria);

                // ahora borro lo de trayectorias anteriores.
                i=Posx+1;
                fseek(Trayectoria,Posx+1,SEEK_SET);
                while(i<MAX_X)
                {
                        fputc(' ',Trayectoria);
                        i++;
                }
                i=Posy+1;
                fseek(Trayectoria,Posy+1,SEEK_SET);
                while(i<2*MAX_X)
                {
                        fputc(' ',Trayectoria);
                        i++;
```

```
                        }
                        i=PosTheta+1;
                        fseek(Trayectoria,PosTheta+1,SEEK_SET);
                        while(i<3*MAX_X)
                        {
                                fputc(' ',Trayectoria);
                                i++;
                        }

                        fclose(Trayectoria);
                        fclose(Proceso);

                // y borro los nodos de la ruta para no confundir con nueva llamada....
                // del mismo modo deberia hacer con TareaActual
                } // Fin del else del fork (odometria)
        } // Fin del else del fork (vision)
        kill (idodom,SIGRTMIN+2); //Finalizo odometria
        kill (idvision,SIGRTMIN); //Finalizo vision
        fin_ekf ();
        printf ("\nFinalice todo correctamente\n");
        /*Fin de la funcion Gestor_Trayecto */
}

void CorrigeTProcesoVision(Posicion *Pos, float *V,float *omega,float *tiempo,int *veces_vision)
{

        int i;

        ////////////////////////////////////
        // Se "corrigen" los datos de vision
        ////////////////////////////////////

        for (i=0;i<*veces_vision+1;i++)
        {
                Pos->x+= tiempo[i]*V[i]*cos(Pos->Theta*(2*PI)/360)/ESCALA;
                Pos->y+= tiempo[i]*V[i]*sin(Pos->Theta*(2*PI)/360)/ESCALA;
                Pos->Theta+= tiempo[i]*omega[i]*360/(2*PI);
                        // omega en rad/seg. Tiene que estar en grados
        }

        if (fabs(Pos->Theta)>=360)
                Pos->Theta=fmod (Pos->Theta,360);

        if (Pos->Theta<0)
                Pos->Theta+=360;

                // Recolocamos los arrays y reinicializamos veces_vision

        V[0]=V[*veces_vision];
        omega[0]=omega[*veces_vision];
        tiempo[0]=tiempo[*veces_vision];

        *veces_vision=0;
}
```

navega.h

```
#include <string.h>
#include <math.h>
#include <stdio.h>
#include <stdlib.h>

#define MAX_NODOS_MAPA    200
#define MAX_LONG_NODO     200
#define MAX_LONG_NOMBRE   3
#define MAX_NODOS_RUTA    50
#define MAX_XC            20000
#define ESCALA            500

void Gestor_Trayecto(char *Destino);
```

navega.c

```
#include "navega.h"
#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>
#include <sched.h>
#include <sys/mman.h>
#include <time.h>
#include <signal.h>
#include <sys/ipc.h>
#include <sys/msg.h>

#include "colas.h"

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;

FILE *Sigueme;
FILE *Movil;
FILE *Consigna;

//-----------------------------------------------------------------
// Nombre: main
// Funcion:
//      Arranca el proceso global de la navegacion.
// Parametros entrada:
//              Inicialmente se proporciona el nodo destino en la
//              llamada al proceso, a falta del proceso de interfaz usuario
// Parametros salida:
//              Solamente se lleva a cabo un unico trayecto, luego la silla
//              se para. No se permite inicilamente cambio de destino en el
//              camino.
//-----------------------------------------------------------------

int colaid;
char *f_mapa;
extern int idneuron;

void manejador (int senal)
{}

void manejador2 (int senal)
{}

int main(int argc, char *argv[])
{
        long i;
        int res;
        pid_t id;

        struct sched_param planif;

        /* Se recoge el nombre del nodo de destino */
        if(argc!=3)
        {
                printf("Error!!! Sintaxis: %s nombre_nodo_destino fichero_mapa \n",argv[0]);
                exit(-1);
        }

        puts("Planificador de Rutas. \nTFC Eduardo González. Navegacion\n\n");
```

```
        /* se supone que tienes el mapa del origen*/
        /* Abriendo el fichero que tiene el mapa */
        if(!(Mapa=fopen(argv[2],"r")))
        {
                perror("Control: Error Abriendo el mapa");
                exit(-1);
        }
        f_mapa=argv[2];

        // Ahora abro un fichero para meter las evolucion del controlador
        if(!(Sigueme=fopen("Sigueme.m","w")))
        {
                puts("Error Abriendo el fichero de controlador");
                exit(-1);
        }
        // preparo el fichero de Matlab.
        fseek(Sigueme,0,SEEK_SET);
        fputs("PosDeseo=[",Sigueme);

        // Ahora abro un fichero para meter las evolucion del movil
        if(!(Movil=fopen("Movil.m","w")))
        {
                puts("Error Abriendo el fichero de controlador");
                exit(-1);
        }
        // preparo el fichero de Matlab.
        fseek(Movil,0,SEEK_SET);
        fputs("PosReal=[",Movil);

        // Ahora abro un fichero para meter las consignas
        if(!(Consigna=fopen("Consigna.m","w")))
        {
                puts("Error Abriendo el fichero de consignas");
                exit(-1);
        }
        // preparo el fichero de Matlab.
        fseek(Consigna,0,SEEK_SET);
        fputs("Consigna=[",Consigna);

        // Inicializaciones de planificacion
        // ***********************************************
        if((colaid = msgget(KEY, IPC_CREAT|0660)) == -1)
                perror("Control: Error al acceder a la cola (msgget)");

        // Ponemos las señales a las que reaccionamos
        signal (SIGRTMIN+4, manejador);

        id=getpid();

// ****************** BLOQUEO MEMORIA ************************************

        res = mlockall (MCL_CURRENT);
        if (res==-1)
        {
                perror ("\nControl:Fallo mlockall con MCL_CURRENT");
                exit (1);
        }
        else printf ("\nControl:Bloqueadas las páginas actuales");

        res = mlockall (MCL_FUTURE);
        if (res==-1)
        {
                perror ("\nControl:Fallo mlockall con MCL_FUTURE");
                munlockall();
                exit (1);
        }
        else printf ("\nControl:Bloqueadas las páginas futuras");

// ****************** PLANIFICACIÓN ************************************

        planif.sched_priority=sched_get_priority_max(SCHED_FIFO);
        // Para "RT", padre, prioridad maxima-1
        res=sched_setscheduler(id,SCHED_FIFO,&planif);
        if (res==-1)
        {
                perror ("\nControl: ERROR al cambiar la política de planificación");
                exit (1);
        }
        else printf ("\nControl: Planificación RR\n");

        //***********************************************
        puts("\n/* Llamando al gestor de trayectorias */");
        Gestor_Trayecto(argv[1]);

// ********************************************************************************
// ****************** FIN DEL PROGRAMA ********************************************
// ********************************************************************************
```

```
        // Cerramos el neuron
        close (idneuron);

        // solo se cierra el mapa cuando se captura otro.
        // como en principio solo hay un mapa se cierra al salir.
        /* y cerramos el fichero mapa */
        fclose(Mapa);

        // acabando con el fichero de Matlab
        fputs("];",Sigueme);
        // ahora borro lo de trayectorias anteriores.
        i=ftell(Sigueme);
        while(i<MAX_XC)
        {
                fputc(' ',Sigueme);
                i++;
        }
        fclose(Sigueme);

        // acabando con el fichero de Matlab
        fputs("];",Movil);
        // ahora borro lo de trayectorias anteriores.
        i=ftell(Movil);
        while(i<MAX_XC)
        {
                fputc(' ',Movil);
                i++;
        }
        fclose(Movil);

        // acabando con el fichero de Matlab
        fputs("];",Consigna);
        // ahora borro lo de trayectorias anteriores.
        i=ftell(Consigna);
        while(i<MAX_XC)
        {
                fputc(' ',Consigna);
                i++;
        }
        fclose(Consigna);

        /* Fin de la funcion principal main */
        return 0;
}
```

planruta.h

```
#include <string.h>
#include <math.h>
#include <stdio.h>
#include <stdlib.h>

#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_2   2
#define PASO_SUBRED1_3   2
#define SALA_SUBRED1_01  1
#define SALA_SUBRED1_2   2
#define SALA_SUBRED1_3   2
#define SALA_SUBRED2_3   3
#define CENTRAL_01       3
#define CENTRAL_2        5
#define INTERIOR2_12     5
#define INTERIOR1_12     2

extern void Buscar_Nodo (int, char *);

extern FILE *Mapa;
extern int Ruta[MAX_NODOS_MAPA];
```

planruta.c

```
static int Origen3,Origen2,Origen1,Destino3,Destino2,Destino1;
static int i=0;
static int NodoOrigen, NodoDestino;
static int ListaNodos[MAX_NODOS_MAPA];
```

```
extern void Buscar_Nodo (int,char*);

/*-----------------------------------------------------------------*/
// 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, y ya veremos si hay que meter uno de 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 en el nivel 2 jerarquico (salas comunes)*/
                        if((Pasillo1==0) && (Pasillo2==0))
                        {       /* Ahora chequeo en funcion de la planta */
                                switch ( Planta1 )/* ... o planta 2.... */
                                {
                                        case 0:
                                        case 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 que estoy desde salas comunes a nodos jerarquicos nivel2  */
                        else if (Pasillo1==0)
                                switch (Planta1)
                                {
```

```
                        case 0:
                        case 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 he de ver si paso de nivel jerarquico a sala comun */
        else if (Pasillo2==0)
                switch (Planta1)
                {
                        case 0:
                        case 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:
                                                case 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[MAX_LONG_NODO];
```

```
int Ascensor;
float Distancia1,Distancia2,Distancia5,Distancia6,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 */
Buscar_Nodo(NodoPartida,Nodo);

/* En la variable "nodo" tengo el nodo cuyo nombre coincide con NodoPartida */
/* Ahora hay que recoger el campo 2º y 3º que son respectivamente la posicion x e y */
strtok (Nodo," ");// dummy
xPartida=((float)atoi(strtok('\0'," ")))/100; /* El segundo campo es x */
yPartida=((float)atoi(strtok('\0'," ")))/100; /* 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;
Buscar_Nodo(Ascensor,Nodo);

/* 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 */
strtok (Nodo," ");// dummy
x1=((float)atoi(strtok('\0'," ")))/100;       /* El segundo campo es x */
y1=((float)atoi(strtok('\0'," ")))/100;       /* 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;
Buscar_Nodo(Ascensor,Nodo);

/* 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 */
strtok (Nodo," ");// dummy
x6=((float)atoi(strtok('\0'," ")))/100;       /* El segundo campo es x */
y6=((float)atoi(strtok('\0'," ")))/100;       /* El tercer campo es y */

/* Hallando la distancia al ascensor 6 */
Distancia6=pow((x6-xPartida),2)+pow((y6-yPartida),2);
Distancia6=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;
        Buscar_Nodo(Ascensor,Nodo);

/* 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 */
        strtok (Nodo," ");// dummy
        x2=((float)atoi(strtok('\0'," ")))/100;       /* El segundo campo es x */
        y2=((float)atoi(strtok('\0'," ")))/100;       /* 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;
        Buscar_Nodo(Ascensor,Nodo);

/* 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 */
        strtok (Nodo," ");// dummy
        x5=((float)atoi(strtok('\0'," ")))/100;       /* El segundo campo es x */
        y5=((float)atoi(strtok('\0'," ")))/100;       /* 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[MAX_LONG_NODO];
//char *pr,prueba[10];

/* Busco el nodo origen */
Buscar_Nodo(NodoOrigen,Nodo);

/* 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 */

strtok (Nodo," ");// dummy
xOrigen=((float)atoi(strtok(NULL," ")))/100;       /* El segundo campo es x */
yOrigen=((float)atoi(strtok('\0'," ")))/100;       /* 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*/
Buscar_Nodo(Origen3*100,Nodo);

/* 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 */
strtok (Nodo," ");// dummy
xMapa=((float)atoi(strtok('\0'," ")))/100;       /* El segundo campo es x */
yMapa=((float)atoi(strtok('\0'," ")))/100;       /* 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 */
Buscar_Nodo(Origen3*100+7,Nodo);

/* 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 */
strtok (Nodo," ");// dummy
xMapa=((float)atoi(strtok('\0'," ")))/100;       /* El segundo campo es x */
yMapa=((float)atoi(strtok('\0'," ")))/100;       /* 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;
        }
        default:
                                return -1;
```

```
                                        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 recibe tratamiento especial porque esta en un 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 recibe tratamiento especial porque esta en un 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++]);
        printf("%d \n", Ruta[i]);
}
```

utilnodo.h

```
#include <string.h>
#include <math.h>
#include <stdio.h>
#include <stdlib.h>

#define MAX_LONG_NODO 200
#define MAX_NODOS_MAPA 200

extern FILE *Mapa;
```

utilnodo.c

```
#include "utilnodo.h"

//---------------------------------------------------------------
// 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
//---------------------------------------------------------------

void Buscar_Nodo(int NombreNodo, char* result)
{
        char Nodo[MAX_LONG_NODO];
        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);

        }while((NombreNodo!=atoi(Nodo))&&((j++)<=MAX_NODOS_MAPA));

        /* En la variable "nodo" tengo el nodo cuyo nombre coincide con nodo_Partida */
        if(j<MAX_NODOS_MAPA)
        {
                strcpy (result,Nodo);
                return;
        }
        else
        {
                puts("Nodo no encontrado");
                result=NULL;
                return;
        }
/* Fin de Funcion Bucar_Nodo */
}
```

vision.h

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>
#include "libmatrices.h"

int Barras(tMatriz barker,tMatriz I,tMatriz nmarca);
void Calibra(tMatriz circulos,tMatriz salidacalib);
void Fbrk(tMatriz I, tMatriz salida1);
int Fcirc(tMatriz circulos,tMatriz salida1,tMatriz nmarcas,tMatriz I);
void Fsubwin(tMatriz C1,tMatriz C2,tMatriz C3,tMatriz C4,tMatriz imagen, tMatriz cluster);
void Posicion(tMatriz puntos,float pantilt[],float camara[],float marca[],float*angulos, float *POS);
int Proceso(tMatriz Imagen);
void Fctr(tMatriz WAna,float umbral,tMatriz centro,float *fili,float *filf);
void Fclst(tMatriz entrada, tMatriz salida);
```

barras.c

```
#include "vision.h"
#include <string.h>

#define umbral  0.1

int Sincron(tMatriz flt2,tMatriz puntos,tMatriz code);
void Decode(tMatriz entrada,tMatriz salida);
int Checksum(tMatriz codigo);
int Porcentaje(tMatriz lista,int *numero);

//-------------------------------------------------------------------------
//      Nombre: Barras.
//      Funcion:
//              Detecta y procesa el codigo de barras dentro de la MPL. Obtiene del
//              codigo el numero de marca.
//      Parametros de entrada:
//              tMatriz barker -> matriz con los datos relativos a la posicion del
//                                                              Barker deuna MLP.
//              tMatriz I -> Imgen a procesar.
//              tMatriz nmarca -> matriz con los datos obtenidos, el formato es el
//                                                              siguiente: [numero_marca codigo].
//      Parametros de salida:
//              Devuelve 1 si se ha procesado el código correctamente o 0 si fue erroneo.
//-------------------------------------------------------------------------

//                  f=1,c=9                  f=1,c=9
int Barras(tMatriz barker,tMatriz I,tMatriz nmarca)
{
        float tst,wide;
        int fila,vacio,j,x,a,pts,lineas,valido,flin,clin;
        //int npts,fila,vacio,i,j,x,a,pts,lineas,valido;
        int p1x,p2x,p1y,p2y,vale,correcto,numero;

        tMatriz linea,flt,flt2,picos,puntos,code,lista,codigo;

        TamanoMatriz(&vacio,&fila,I);
        TamanoMatriz(&flin,&clin,I);
        linea=CrearMatriz(vacio,fila);

        //Creamos subventana del codigo de barras para cada marca existente
        //dentro de la MPL
        wide=barker.datos[0][1]-barker.datos[0][0]+1;
        p1x=Redondeo(0.167*wide+barker.datos[0][1]);
        p2x=Redondeo(0.833*wide+p1x);
        p1y=Redondeo(barker.datos[0][2])-5;
        p2y=Redondeo(barker.datos[0][3])+10;

        codigo=CrearMatriz(4,2);
        code=CrearMatriz(1,35); //35 bits de datos tiene un codigo de barras
        lista=CrearMatriz(1,p2x-p1x+1);
        lineas=0;
        for(x=p1x-1;x<p2x;x++)
        {
                for (a=x;a<flin;a++)
                        memcpy (&linea.datos[a-x][0],&I.datos[a][0],clin*sizeof(float));

                flt=CrearMatriz(1,fila);
                //Se aplica un filtro+derivada para eliminar ruido aleatorio de
                //fondo y detectar transiciones bruscas.
                CerosMatriz(1,fila,flt);
                //Cargamos los datos normalizados
                for(a=1;a<fila-1;a++)
                {
                        flt.datos[0][a]=linea.datos[0][a+1]/255.0-linea.datos[0][a-1]/255.0;
                }
                flt2=CrearMatriz(1,fila);
                j=0;
                for(a=p1y-1;a<p1y+p2y-1;a++)
                {
                        flt2.datos[0][j]=flt.datos[0][a];
                        j++;
                }
                *flt2.col_real=j;
                picos=CrearMatriz(1,*flt2.col_real);
                CerosMatriz(1,*flt2.col_real,picos);
                //Buscamos los picos de flt2 (tanto positivos como negativos)
                for (a=1;a<*flt2.col_real-1;a++)
                {
                        tst=flt2.datos[0][a];
                        if (tst>umbral)
                        {
                                picos.datos[0][a]=(flt2.datos[0][a-1]<=tst)&
                                                  (tst>flt2.datos[0][a+1]);
                        }
                        else if(tst<-umbral)
                        {
                                picos.datos[0][a]=-((flt2.datos[0][a-1]>=tst)&
```

```
                                                (tst<flt2.datos[0][a+1]));
                        }
                }
                //Deteccion y registro de los picos (maximos locales)
                puntos=CrearMatriz(1,*flt2.col_real);
                Buscar(picos,puntos,'!',0);
                //Registro el numero de flancos detectados en 'npts'
                TamanoMatriz(&vacio,&pts,puntos);
                if (pts>20)
                {
                        vale=Sincron(flt2,puntos,code);
                        if (vale==1)
                        {
                                Decode(code,codigo);
                                valido=Checksum(codigo);
                                if (valido==1)
                                {
                                        lista.datos[0][lineas]=0;
                                        for(j=0;j<4;j++)
                                        {
                                                lista.datos[0][lineas]=lista.datos[0][lineas]+
                                                                        codigo.datos[j][1]*
                                                                        pow(10,3-j);
                                        }
                                        lineas++;//numero de lineas detectadas pasando el checksum

                                }
                        }
                }
                LiberarMatriz(flt);
                LiberarMatriz(flt2);
                LiberarMatriz(picos);
                LiberarMatriz(puntos);
        }
        *lista.col_real=lineas;
        correcto=Porcentaje(lista,&numero);
        if (correcto==1)
        {
                nmarca.datos[0][0]=0;nmarca.datos[0][1]=numero;
                printf("\nCodigo de barras correcto: %d\n",numero);
        }
        else printf("\nError leyendo codigo de barras\n");
        LiberarMatriz(codigo);
        LiberarMatriz(code);
        LiberarMatriz(lista);
        LiberarMatriz(linea);
        return correcto;
}

//-------------------------------------------------------------------------
//      Nombre: Sincron.
//      Funcion:
//              Obtienen el eperiodo de bit a a partir de las barras de sincronismo y
//              ademas recupera la secuencia completa de 0's y 1's
//      Parametros de entrada:
//              tMatriz flt2 -> matriz con los datos normalizados del codigo.
//              tMatriz puntos -> matriz que contiene el numero de flancos.
//              tMatriz code -> matriz con los datos obtenidos, correspondientes a la
//                                                      secuencia recuperada.
//      Parametros de salida:
//              int vale -> valida el codigo
//-------------------------------------------------------------------------
int Sincron(tMatriz flt2,tMatriz puntos,tMatriz code)
{
        float umbralT=0.028571,*T;
        int npts,pts,aux,x,i,j,indice,bit,vale;
        tMatriz T1;

        T1=CrearMatriz(1,5);

        T=(float *)malloc(5*sizeof(float));

        TamanoMatriz(&aux,&npts,puntos);
        TamanoMatriz(&aux,&pts,flt2);
        //Ponderamos los periodos obtenidos del sincronismo.
        T1.datos[0][0]=puntos.datos[0][1]-puntos.datos[0][0];
        T1.datos[0][1]=puntos.datos[0][3]-puntos.datos[0][2];
        T1.datos[0][2]=puntos.datos[0][npts-1]-puntos.datos[0][npts-2];
        T1.datos[0][3]=puntos.datos[0][npts-3]-puntos.datos[0][npts-4];
        T1.datos[0][4]=puntos.datos[0][npts-5]-puntos.datos[0][npts-6];
        Media(T1,2,T);

        if (ceil(*T)>=Redondeo(pts*umbralT))
        {
                UnosMatriz(1,35,code);
                x=0;
                for(i=3;i<=npts-6;i++)   //lectura codigo
```

```
                {
                        bit=Redondeo((puntos.datos[0][i+1]-puntos.datos[0][i])/ceil(*T));
                        indice=puntos.datos[0][i];
                        if (flt2.datos[0][indice]<0)
                        {
                                for(j=x;j<=x+bit-1;j++)
                                {
                                        code.datos[0][j]=0;
                                }
                                x=x+bit;
                        }
                        else
                        {
                                x=x+bit;
                        }
                }
                vale=1;
        }
        else
        {
                vale=0;
        }
        LiberarMatriz(T1);
        free (T);
        return(vale);
}
//---------------------------------------------------------------------------
//      Nombre: Decode.
//      Funcion:
//              Decodifica la secuencia recuperada.
//      Parametros de entrada:
//              tMatriz entrada -> matriz con los datos correspondientes a la secuencia
//                                                              recuperada.
//              tMatriz salida -> matriz con el valor decodificado.
//      Parametros de salida:
//     Ninguno, ya que son pasados por referencia.
//---------------------------------------------------------------------------
void Decode(tMatriz entrada,tMatriz salida)
{
        int i,j,ind=0;
        float valor;
        tMatriz tabla,x,bit;

        x=CrearMatriz(1,1);
        bit=CrearMatriz(1,7);
        tabla=CrearMatriz(1,20);

        x.datos[0][0]=0;
        tabla.datos[0][0]=1110010;tabla.datos[0][1]=1100110;tabla.datos[0][2]=1101100;
        tabla.datos[0][3]=1000010;tabla.datos[0][4]=1011100;tabla.datos[0][5]=1001110;
        tabla.datos[0][6]=1010000;tabla.datos[0][7]=1000100;tabla.datos[0][8]=1001000;
        tabla.datos[0][9]=1110100;tabla.datos[0][10]=1011000;tabla.datos[0][11]=1001100;
        tabla.datos[0][12]=1100100;tabla.datos[0][13]=1011110;tabla.datos[0][14]=1100010;
        tabla.datos[0][15]=1000110;tabla.datos[0][16]=1111010;tabla.datos[0][17]=1101110;
        tabla.datos[0][18]=1110110;tabla.datos[0][19]=1101000;
        for(i=0;i<=21;i=i+7)
        {
                memcpy (&bit.datos[0][0],&entrada.datos[0][i],7*sizeof(float));
                valor=0;
                for(j=0;j<7;j++)
                {
                        valor=valor+bit.datos[0][6-j]*pow(10,j);
                }
                Buscar(tabla,x,'=',valor);
                if (x.datos[0][0]>10)
                {
                        salida.datos[ind][0]=1;salida.datos[ind][1]=x.datos[0][0]-10;
                        ind++;
                }
                else if (x.datos[0][0]<=10)
                {
                        salida.datos[ind][0]=0;salida.datos[ind][1]=x.datos[0][0];
                        ind++;
                }
        }
        LiberarMatriz(tabla);
        LiberarMatriz(x);
        LiberarMatriz(bit);

}
//---------------------------------------------------------------------------
//      Nombre: Checksum.
//      Funcion:
//              Comprueba el valor decodificado, verificando el checksum
//      Parametros de entrada:
//              tMatriz codigo -> matriz con los valores decodificados
//      Parametros de salida:
```

```
//              int vale -> valida el codigo
//-------------------------------------------------------------------------
int Checksum(tMatriz codigo)
{
        int i,npts,aux,check=0,suma=0,resto,vale;
        TamanoMatriz(&npts,&aux,codigo);
        if (npts==4)
        {
                for(i=0;i<4;i++)
                {
                        check=check+pow(2,3-i)*(codigo.datos[i][0]);
                        suma=suma+codigo.datos[i][1];
                }
                while (suma>=10)
                {
                        resto=suma-10*floor(suma/10);
                        suma=resto;
                }
                if (resto==check)
                {
                        vale=1;
                }
                else
                {
                        vale=0;
                }
        }
        else
        {
                vale=0;
        }
        return(vale);
}
//-------------------------------------------------------------------------
//      Nombre: Porcentaje.
//      Funcion:
//              Comprueba que el numero de codigos recuperados supera el 50% de las
//              lineas exploradas
//      Parametros de entrada:
//              tMatriz lista -> matriz con los valores recuperados para cada linea
//                                                              del codigo leida.
//              int *numero -> retorna el numero final perteneciente al codigo leido
//      Parametros de salida:
//              int vale ->valida el codigo
//-------------------------------------------------------------------------
int Porcentaje(tMatriz lista,int *numero)
{
        int s=1,i,x=0,vacio,lineas,vale;
        float aux;
        tMatriz matriz,salida;
        TamanoMatriz(&vacio,&lineas,lista);
        vacio=lineas;
        matriz=CrearMatriz(1,lineas+1);
        salida=CrearMatriz(1,lineas+1);
        *numero=9999;
        //ordenamos los valores dentro de lista
        if (lineas>0)
        {
                while((s==1)&&(--lineas))
                {
                        s=0;
                        for(i=1;i<lineas;i++)
                        {
                                if(lista.datos[0][i]>=lista.datos[0][i-1])
                                {
                                        aux=lista.datos[0][i-1];
                                        lista.datos[0][i-1]=lista.datos[0][i];
                                        lista.datos[0][i]=aux;
                                        s=1;
                                }
                        }
                }
                for(i=0;i<vacio-1;i++)
                {
                        if (lista.datos[0][i]!=lista.datos[0][i+1])
                        {
                                matriz.datos[0][x]=lista.datos[0][i+1];
                                x++;
                        }
                }
                matriz.datos[0][x]=lista.datos[0][0];
                for(i=0;i<=x;i++)
                {
                        Buscar(lista,salida,'=',matriz.datos[0][i]);
                        if (*salida.col_real>0.5*vacio)
                        {
                                *numero=(int)matriz.datos[0][i];
```

```
                                vale=1;
                        }
                }
        }
        else
        {
                vale=0;
        }
        LiberarMatriz(matriz);
        LiberarMatriz(salida);
        return(vale);
}
```

calibra.c

```
#include "vision.h"

typedef struct
{
        float fc[2];
        float cc[2];
        float alpha_c;
        int numelem;
        float kc[5];
}Estruc_calib;

void Normalize(tMatriz x_kk,float fc[],float cc[],float kc[],int num,
                                        float alpha_c,tMatriz xn);
void Comp_distortion_oulu(tMatriz xd, float k[], int num, tMatriz x);
void Comp_distortion(tMatriz x_dist, float k2[], int num, tMatriz x_comp);

//-------------------------------------------------------------------------
//      Nombre: Calibra.
//      Funcion:
//              Realiza la recalibracion de los centroides de cada marca en funcion
//              de los parametros de calibracion contenidos en el fichero Calib.txt.
//      Parametros de entrada:
//              tMatriz circulos -> parametros para hacer la recalibracion.
//              tMatriz salidacalib -> centroides ya rectificados.
//      Parametros de salida:
//              Ninguno, ya que son pasados por referencia.
//-------------------------------------------------------------------------
void Calibra(tMatriz circulos,tMatriz salidacalib)
{
        FILE *Calib;
        Estruc_calib datos;
        tMatriz x_kk, media, xn;
        float *med;
        int nf,nc,i,j,x,y;
        x_kk=CrearMatriz(2,4);
        media=CrearMatriz(2,4);
        xn=CrearMatriz(2,4);
        //se cargan los parametros de calibracion de la camara
        if((Calib=fopen("Calib.txt", "rb"))==NULL)
        {
                perror("ERROR: ");
                exit(0);
        }

        rewind(Calib);
        fread(&datos,sizeof(Estruc_calib),1,Calib);
        if (ferror(Calib))
                printf("\nError fichero de calibracion\n");
        fclose(Calib);

        med=(float *)malloc(4*sizeof(float));
        //ejecutamos el proceso tantas veces como marcas contenga
        //la variable circulos

        TamanoMatriz(&nf,&nc,circulos);

        for(i=0;i<nf;i++)
        {
                for(j=0;j<4;j++)
                {
                        x_kk.datos[0][j]=circulos.datos[i][2*j];
                        x_kk.datos[1][j]=circulos.datos[i][2*j+1];
                }
                //los centroides de cada elipse estan en circulos.
                //se rectifica la posicion de esos puntos y se obtiene salidacalib.

                Normalize(x_kk,datos.fc,datos.cc,datos.kc,datos.numelem,datos.alpha_c,xn);
                //desplazamos el origen de todos los puntos 'xn' para eliminar el
                //efecto del offset en las posiciones del MLP en las imagenes
                Media(xn,2,med);
                for(j=0;j<4;j++)
                {
```

```
                        media.datos[0][j]=med[0];
                        media.datos[1][j]=med[1];
                }
                for(x=0;x<2;x++)
                {
                        for(y=0;y<4;y++)
                        {
                                xn.datos[x][y]=xn.datos[x][y]-media.datos[x][y];
                        }
                }
                for(j=0;j<4;j++)
                {
                        salidacalib.datos[i][2*j]=xn.datos[0][j];
                        salidacalib.datos[i][2*j+1]=xn.datos[1][j];
                }
                salidacalib.datos[i][8]=circulos.datos[i][8];
        }
        *salidacalib.fil_real=nf;
        LiberarMatriz(x_kk);
        LiberarMatriz(media);
        LiberarMatriz(xn);
        free (med);
}
//-------------------------------------------------------------------------
//      Nombre: Normalize.
//      Funcion:
//              Obtiene las coordenadas normalizadas xn en funcion de las coordenadas
//              en pixels y de los parametros intrinsecos de la camara fc, cc y kc.
//      Parametros de entrada:
//              tMatriz x_kk -> coordenadas de los centroides de la imagen.
//              float fc[] -> longitud focal de la camara.
//              float cc[] -> punto principal de coordenada.
//              float kc[] -> coeficientes de distorsion.
//              int num -> numero de coeficientes de distorsion.
//              float alpha_c -> coeficiente de oblicuidad.
//              tMatriz xn -> coordenadas normalizadas.
//      Parametri¡os de salida:
//              Ninguno, ya que son pasados por referencia.
//-------------------------------------------------------------------------
void Normalize(tMatriz x_kk,float fc[],float cc[],float kc[],int num,
                                        float alpha_c,tMatriz xn)
{
        tMatriz x_distor;
        int i,j;

        x_distor=CrearMatriz(2,4);
        //Primero: se sustrae el punto principal y se divide por la longitud focal:

        for(i=0;i<2;i++)
        {
                for(j=0;j<4;j++)
                {
                        x_distor.datos[i][j]=(x_kk.datos[i][j]-cc[i])/fc[i];
                }
        }
        //Segundo: eliminamos la oblicuidad.

        for(j=0;j<4;j++)
        {
                x_distor.datos[0][j]=x_distor.datos[0][j]-alpha_c*x_distor.datos[1][j];
        }

        //Tercero: si el mayor valor singular del vector kc es distinto de cero
        //realizamos una compensacion de la distorcion producida por la lente.
        if(MVS(kc,num)!=0)
        {
                Comp_distortion_oulu(x_distor,kc,num,xn);
        }
        else
        {
                for(i=0;i<2;i++)
                {
                        memcpy (&xn.datos[i][0],&x_distor.datos[i][0],4*sizeof(float));
                        /*for(j=0;j<4;j++)
                        {
                                xn.datos[i][j]=x_distor.datos[i][j];
                        }*/
                }
        }
        LiberarMatriz(x_distor);
}
//-------------------------------------------------------------------------
//      Nombre: Comp_distortion_oulu.
//      Funcion:
//              Compensa la distorsion radial y tangencial. Usando metodo iterativo por
//              compensacion.
//      Parametros de entrada:
```

```
//              tMatriz xd -> coordenadas distorsionadas en el plano imagen.
//              float k[] -> coeficientes de distorsion.
//              int num -> numero de coeficientes.
//              tMatriz x -> coordenadas no distorsionadas del plano imagen.
//      Parametros de salida:
//              Ninguno , ya que son pasados por referencia.
//-----------------------------------------------------------------------
void Comp_distortion_oulu(tMatriz xd, float k[], int num, tMatriz x)
{
        int i,j,kk;
        tMatriz r_2, k_radial, delta_x;
        float k1,k2,k3,p1,p2;
        r_2=CrearMatriz(1,4);
        k_radial=CrearMatriz(1,4);
        delta_x=CrearMatriz(2,4);

        if (num==1)
        {
                Comp_distortion(xd,k,num,x);
        }
        else
        {
                //k contiene 5 coeficientes de distorsion.

                k1=k[0];
                k2=k[1];
                k3=k[2];
                p1=k[3];
                p2=k[4];

                for(i=0;i<2;i++)
                {
                        memcpy (&x.datos[i][0],&xd.datos[i][0],4*sizeof(float));
                        /*for(j=0;j<4;j++)
                        {
                                x.datos[i][j]=xd.datos[i][j];
                        }*/
                }

                for(kk=0;kk<5;kk++)
                {
                        for(i=0;i<2;i++)
                        {
                                for(j=0;j<4;j++)
                                {
                                        x.datos[i][j]=(float)pow((double)x.datos[i][j],2);
                                }
                        }

                        Sum(x,r_2,1);

                        for(j=0;j<4;j++)
                        {

        k_radial.datos[0][j]=1+k1*r_2.datos[0][j]+k2*(float)pow((double)r_2.datos[0][j],2)+
                                k3*(float)pow((double)r_2.datos[0][j],3);

        delta_x.datos[0][j]=2*p1*x.datos[0][j]*x.datos[1][j]+p2*(r_2.datos[0][j]+2*
                                (float)pow((double)x.datos[0][j],2));

        delta_x.datos[1][j]=p1*(r_2.datos[0][j]+2*(float)pow((double)x.datos[1][j],2))+
                                2*p2*x.datos[0][j]*x.datos[1][j];
        x.datos[0][j]=(xd.datos[0][j]-delta_x.datos[0][j])/k_radial.datos[0][j];
        x.datos[1][j]=(xd.datos[1][j]-delta_x.datos[1][j])/k_radial.datos[0][j];
                        }

                }
        }
        LiberarMatriz(r_2);
        LiberarMatriz(k_radial);
        LiberarMatriz(delta_x);
}
//-----------------------------------------------------------------------
//      Nombre: Comp_distortion.
//      Funcion:
//              Realiza sobre la imagen una compensacion de la distorsion radial
//              de la camara.
//      Parametros de entrada:
//              tMatriz x_dist -> matriz con los puntos obtenidos sin considerar la
//                                                              distorsion radial.
//              float k2[] -> coeficiente(s) de distorsion.
//              int num -> numero de coeficientes.
//              tMatriz x_comp -> matriz con los puntos ya corregidos despues del
//                                                              efecto de la distorsion.
//      Parametros de salida:
//              Ninguno, ya que son pasados por referencia.
//-----------------------------------------------------------------------
```

```
void Comp_distortion(tMatriz x_dist, float k2[], int num, tMatriz x_comp)
{
        int two,N,j;
        tMatriz radius_2, radial_distortion, radius_2_comp;
        radius_2=CrearMatriz(1,4);
        radial_distortion=CrearMatriz(2,4);
        radius_2_comp=CrearMatriz(1,4);

        TamanoMatriz(&two,&N,x_dist);
        if (two!=2)
        {
                printf("ERROR: La dimension debe ser 2x1\n");
                exit(0);
        }
        if (num>1)
        {
                Comp_distortion_oulu(x_dist,k2,num,x_comp);
        }
        else
        {
                for(j=0;j<4;j++)
                {
                        radius_2.datos[0][j]=(float)pow((double)x_dist.datos[0][j],2)+
                        (float)pow((double)x_dist.datos[1][j],2);
                        radial_distortion.datos[0][j]=1+k2[0]*radius_2.datos[0][j];
                        radial_distortion.datos[1][j]=1+k2[0]*radius_2.datos[0][j];
                        radius_2_comp.datos[0][j]=((float)pow((double)x_dist.datos[0][j],2)+
                        (float)pow((double)x_dist.datos[1][j],2))/radial_distortion.datos[0][j];
                        radial_distortion.datos[0][j]=1+k2[0]*radius_2_comp.datos[0][j];
                        radial_distortion.datos[1][j]=1+k2[0]*radius_2_comp.datos[0][j];
                        x_comp.datos[0][j]=x_dist.datos[0][j]/radial_distortion.datos[0][j];
                        x_comp.datos[1][j]=x_dist.datos[1][j]/radial_distortion.datos[1][j];
                }
        }
        LiberarMatriz(radius_2);
        LiberarMatriz(radial_distortion);
        LiberarMatriz(radius_2_comp);
}
```

captura.c

```
#define CAMARA                  "/dev/video0"
#define IMAG_ANCHO              640
#define IMAG_ALTO               480
#define NUM_PIXELS              IMAG_ANCHO*IMAG_ALTO
#define X_DEPTH                 8
#define FORMATO_PIXEL           V4L2_PIX_FMT_GREY
#define INDICE_S_VIDEO          2
#define KEY                     31416   // Para la cola de mensajes
#define NUM_REINTENTOS          10

#include <sys/time.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/ioctl.h>
#include <fcntl.h>
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <sys/mman.h>
#include <errno.h>

#include <linux/fs.h>
#include <linux/kernel.h>
#include <linux/videodev2.h>  /* Video for Linux Two */

#include <signal.h>      // Recepcion de señales
#include <sched.h>       // Planificacion
#include "vision.h"
#include "colas.h"

#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>

int             vid;
int             x_depth;

tMatriz Imagen;
int colaid;
FILE *id_fmapa;
char letra_mapa;

int Iteracion();
```

```
void manejador ()
{
        int err;

        close(vid);
        fclose(id_fmapa);

        err = munlockall ();
        if (err==-1)
                perror ("\nVision:Fallo munlockall");
        else printf ("\nVision:Páginas desbloqueadas\n");

        LiberarMatriz(Imagen);
        exit (0);       // Fin del programa
}

void manejador2 ()
{
}       // No hace nada, es sólo para despertar cuando se reciba la señal

int init_video(void)
{
        int     err, index;

        struct v4l2_input input;
        struct v4l2_format fmt;

        vid = open(CAMARA, O_RDWR);       //abro el dispositivo

        //*********************************************************
        //              Configuro el driver y la camara           //
        //*********************************************************
        index=INDICE_S_VIDEO;    // Entrada de imagen que vamos a usar (S-Video)
        if (-1 == ioctl (vid, VIDIOC_S_INPUT, &index))
        {
        perror ("VIDIOC_S_INPUT");
        exit (EXIT_FAILURE);
        }

        input.index = index;

        if (-1 == ioctl (vid, VIDIOC_ENUMINPUT, &input))
        {
                perror ("VIDIOC_ENUMINPUT");
                exit (EXIT_FAILURE);
        }

        fmt.type=V4L2_BUF_TYPE_VIDEO_CAPTURE;
        fmt.fmt.pix.width = IMAG_ANCHO;           //caracteristicas dl formato d frame
        fmt.fmt.pix.height = IMAG_ALTO;           //para adquirirlo
        fmt.fmt.pix.pixelformat = FORMATO_PIXEL;
        fmt.fmt.pix.field = V4L2_FIELD_INTERLACED;

        err = ioctl(vid, VIDIOC_S_FMT, &fmt);
        if (err)
        {
                printf("S_FMT returned error %d\n",errno);
                return 1;
        }

        return 0;
}

int main(int argc, char *argv[])
{
        int err;
        struct sched_param planif;
        pid_t id;

        letra_mapa='-';          // Fuerzo cambio de mapa en 1a iteracion
        id_fmapa=NULL;

        //****************** PLANIFICACIÓN ************************************

        id=getpid();
        planif.sched_priority=0;
        // Hijo 2, prioridad 0 para planificacion normal

        err=sched_setscheduler(id,SCHED_OTHER,&planif);
        if (err==-1)
        {
                perror ("\nVision: ERROR al cambiar la política de planificación");
                exit (1);
        }
        else printf ("\nVision: Planificación NORMAL");
        signal (SIGRTMIN, manejador);     // Usaremos esta señal para que nos indiquen que acabamos
        signal (SIGRTMIN+1, manejador2);  // Usaremos esta señal para que nos indiquen que empecemos
```

```
                                                // una iteracion de vision

        if((colaid = msgget(KEY, IPC_CREAT|0660)) == -1)
        {
                perror("Vision: No puedo acceder a la cola de mensajes (msgget)");
                exit (1);
        }
        //****************** FIN PLANIFICACIÓN ***************************************

        //****************** BLOQUEO MEMORIA *****************************************

        err = mlockall (MCL_CURRENT);
        if (err==-1)
        {
                perror ("\nVision: Fallo mlockall con MCL_CURRENT");
                exit (1);
        }
        else printf ("\nVision:Bloqueadas las páginas actuales");

        err = mlockall (MCL_FUTURE);
        if (err==-1)
        {
                perror ("\nVision:Fallo mlockall con MCL_FUTURE");
                err = munlockall ();
                if (err==-1)
                {
                        perror ("\nVision:Fallo munlockall");
                        exit (1);
                }
                else printf ("\nVision:Páginas desbloqueadas");
                exit (1);
        }
        else printf ("\nVision:Bloqueadas las páginas futuras");

        //****************** FIN BLOQUEO MEMORIA *************************************

        Imagen=CrearMatriz(IMAG_ANCHO, IMAG_ALTO);

        if (init_video())        //empieza a capturar
        {
                perror ("\nError: al configurar el video\n");
                exit (1);
        }

        err=NUM_REINTENTOS+1;

        while (err)      // La primera tiene que tener exito siempre
        {
                        // Primera vez (no hay que hacer sleep entre iteraciones)
                if (!Iteracion())
                        break;
                err--;
        }
        if (!err)        // si llegamos a 0 se acaba
                manejador();

        for (;;)
        {
                Iteracion();

                // Dormimos hasta que se reciba alguna señal (empezar iteracion o salir)
                sleep (5);       // Cinco segs sera suficiente, siempre despertará antes
        }

        /* Salida del programa */

        return 0;        // Aqui no se llega nunca
}

int Iteracion()
{
        int err;
        int i, j;

        char imagen[NUM_PIXELS];

        struct buf_msg_map paq_mapa;
        struct buf_msg_vi paquete;        // Paquete a enviar en caso de error

        err=msgrcv (colaid, &paq_mapa, MAX_SEND_SIZE, MSG_MAPA, IPC_NOWAIT);

        if (err!=-1)
        {
                if (id_fmapa!=NULL)
                        fclose(id_fmapa);

                letra_mapa=paq_mapa.edificio;
```

```
                switch (letra_mapa)
                {
                        case 'N':
                                        id_fmapa=fopen("EpNorte.map","r");
                                        break;
                        case 'S':
                                        id_fmapa=fopen("EpSur.map","r");
                                        break;
                        case 'E':
                                        id_fmapa=fopen("EpEste.map","r");
                                        break;
                        default: // Edificio oeste por defecto*/
                                        id_fmapa=fopen("EpOeste.map","r");
                                        letra_mapa='O';
                                        break;
                }

                if(id_fmapa==NULL)
                {
                        perror ("Vision: Error Abriendo el mapa");
                        exit(-1);
                }
        }

        while (1)
        {       // El bucle se romperá cuando se procese bien una imagen
                reintentos--;
                if (!reintentos)
                {
                        LiberarMatriz(Imagen);

                        printf ("\nError: no se pudo procesar ninguna imagen\n");
                        close (vid);
                        exit (1);
                }*/

                err=read (vid,imagen,NUM_PIXELS); // Leo un frame

                //**********************************************************************
                for(i=0;i<IMAG_ALTO;i++)
                        for(j=0;j<IMAG_ANCHO;j++)
                                Imagen.datos[j][i]=(unsigned char)imagen[j+i*IMAG_ANCHO];
                //**********************************************************************

                if (err==-1)
                {
                        printf ("\nError: Captura de imagen\n");
                        return 1;
                }


                if(!Proceso(Imagen))
                {
                        printf("\nError:se repite iteración de visión\n");
                        fflush (stdout);

                        paquete.mtype=MSG_VISION; // Paquete vision
                        strcpy(paquete.nodo,"---");
                        paquete.datos[0]=0;
                        paquete.datos[1]=0;
                        paquete.datos[2]=0;

                        err=msgsnd(colaid,(struct msgbuf *)&paquete,
                                   sizeof(paquete)-sizeof(long), IPC_NOWAIT);

                        if (err==-1)
                                perror ("\nVision: Error al enviar mensaje");

                        return 1;
                }

        return 0;
}
```

fbrk.c

```
#include "vision.h"

void Flinmx(float *candidato,float *longitud,float *valor,int *idxc,tMatriz linea);
void Fdeltat(tMatriz secuencia,int *valet,float *distancia,tMatriz brkcode);
void Tstrozomx(tMatriz trozo,float *distancia,tMatriz brkcode,int *vale,float *valor);
void Incluye(tMatriz matriz,tMatriz dato,int *idx,int talla);

//---------------------------------------------------------------------------
```

```
//      Nombre: Fbrk.
//      Funcion:
//              Deteccion de codigos Barker-7.Procesa linea a linea detectando un posible
//              codigo Barker.
//      Parametros de entrada:
//              tMatriz I -> Imagen a procesar.
//              tMatriz salida -> matriz que contiene las lineas en las que se ha detectado
//                                                                      un codigo Barker.
//      Parametros de salida:
//              Ninguno, ya que son pasados por referencia.
//-------------------------------------------------------------------------
void Fbrk(tMatriz I, tMatriz salida1)
{
        //Declaracion de tipos
        int lmax, xmax;
        int x,i,p,y,j;
        int aux, nfil, npts;
        int k,valet,vale,idxc,idxs;
        int sum1,sum2,sum3,sum4,sum5,sum6;
        int valido,exito,antcol;
        float distancia,datos,pto,candidato;
        float tst,umbral,umbraln,posible;
        tMatriz flt,flt2,auxiliar,picos,puntos,signo;
        tMatriz secuencia,trozo,brkcode,salida;
        tMatriz linea,dato;
        float xp=0, valor, longitud;

        //Generamos las matrices de tamaño imagen
        TamanoMatriz(&xmax,&lmax,I);
        linea=CrearMatriz(1,lmax);
        dato=CrearMatriz(1,4);
        brkcode=CrearMatriz(1,6);
        puntos=CrearMatriz(1,lmax);
        flt=CrearMatriz(1,lmax);
        flt2=CrearMatriz(1,lmax);
        auxiliar=CrearMatriz(1,1);
        picos=CrearMatriz(1,lmax);
        secuencia=CrearMatriz(1,6);
        trozo=CrearMatriz(1,lmax);
        salida=CrearMatriz(200,9);
        idxs=0;  //esto equivale a matriz vacia ...
        valido=1;
        exito=0;
        antcol=0;
        for(x=0;x<640;x+=2)      //Exploracion de lineas: de '2' en '2'.
        {
                idxc=0;
                if(x==xp-1)
                {
                        x=x+1;
                }
                if((exito==1)&&(antcol==1)&&(valido==0))
                {
                        valido=1;
                }
                else if((exito==1)&&(antcol==0)&&(valido==0))
                {
                        exito=0;
                        valido=1,
                        antcol=0;
                }

                memcpy (&linea.datos[0][0],&I.datos[x][0],lmax*sizeof(float));

                //La funcion Flinmx detecta la presencia de codigos Barker:
                //Buscar posibles candidatos para ser codigos Barker validos.
                //Extrae flancos de la linea dada y comprueba la secuencia
                //adecuada de signos de la derivada: [-,+,-,+,-,+]
                //Se utiliza una aproximacion Sobel de la derivada y un umbral
                //basado en un factor de cresta sobre el valor RMS de la derivada.

                CerosMatriz(1,lmax,flt);
                CerosMatriz(1,lmax,flt2);
                for(i=1;i<lmax-1;i++)
                {
                        flt.datos[0][i]=linea.datos[0][i+1]-linea.datos[0][i-1];
                        flt2.datos[0][i]=flt.datos[0][i]*flt.datos[0][i];
                }


                //El umbral se obtiene como dos y media veces el valor RMS
                //esto es: con un factor de cresta igual a 2.
                Sum(flt2,auxiliar,2);
                umbral=2*(auxiliar.datos[0][0]/lmax);
                umbraln=sqrt(umbral);

                CerosMatriz(1,lmax,picos);
```

```
	//buscamos los picos de flt (tanto positivos como negativos)
	for(i=1;i<lmax-1;i++)
	{
		tst=flt.datos[0][i];
		if (tst>umbraln)
		{
			picos.datos[0][i]=(flt.datos[0][i-1]<=tst)&(tst>flt.datos[0][i+1]);
		}
		if (tst<-umbraln)
		{
			picos.datos[0][i]=-((flt.datos[0][i-1]>=tst)&
					(tst<flt.datos[0][i+1]));
		}
	}
	//Deteccion y registro de los picos (maximos locales)

	Buscar(picos,puntos,'!',0);

	//Registro el numero de flancos detectados en 'npts'
	TamanoMatriz(&aux,&npts,puntos);

	//Filtrado final de los puntos hallados. Buscamos aquellos en
	//los que los signos son los correctos, esto es:
	//la secuencia seria -> [-,+,-,+,-,+]

	if (npts>=6)
	{
		//solo si hay suficientes puntos, se comprueban los signos
		TamanoMatriz(&aux,&nfil,puntos);
		signo=CrearMatriz(1,nfil);
		UnosMatriz(1,nfil,signo);
		for(i=0;i<npts;i++)
		{
			p=(int)puntos.datos[0][i];
			tst=flt.datos[0][p];
			if (tst<0)
			{
				signo.datos[0][i]=-1;
			}
		}
		//en este otro paso verificamos las variaciones
		posible=0;
		for(i=0;i<npts-5;i++)
		{
			if(signo.datos[0][i]==-1)
			{

				sum1=-1*signo.datos[0][i];
				sum2=1*signo.datos[0][i+1];
				sum3=-1*signo.datos[0][i+2];
				sum4=1*signo.datos[0][i+3];
				sum5=-1*signo.datos[0][i+4];
				sum6=1*signo.datos[0][i+5];
				posible=sum1+sum2+sum3+sum4+sum5+sum6;
				if (posible>=6)
				{
					//ahora investigamos si hay un Barker7
					//primero extraemos el trozo de linea:

					for(k=0;k<=5;k++)
					{
					secuencia.datos[0][k]=puntos.datos[0][i+k];
					}
					k=0;

for(y=(int)(secuencia.datos[0][0]);y<=(int)secuencia.datos[0][5];y++)
					{
						trozo.datos[0][k]=linea.datos[0][y];
						k++;
					}
					*trozo.col_real=k;

					Fdeltat(secuencia,&valet,&distancia,brkcode);
					if (valet==1)
					{
						TamanoMatriz(&aux,&nfil,brkcode);
						for(k=0;k<nfil;k++)
						{
						brkcode.datos[0][k]=brkcode.datos[0][k]+1;
						}
					Tstrozomx(trozo,&distancia,brkcode,&vale,&datos);
						if (vale==1)
						{
							pto=puntos.datos[0][i];
							candidato=pto+1;
						//le sumamos 1 para obtener fila absoluta
```

```
                                                                longitud=distancia;
                                                                valor=datos;
                                                                idxc=1;  //indice de dato valido
                                                                break;
                                                                //pasamos a la siguiente linea
                                                        }
                                                        else
                                                        {
                                                                valido=0;
                                                        }
                                                }
                                                else
                                                {
                                                        valido=0;
                                                }
                                        }
                                        else
                                        {
                                                valido=0;
                                        }
                                }
                                else
                                {
                                        valido=0;
                                }
                        }
                        LiberarMatriz(signo);
                }
                else
                {
                        valido=0;
                }

                //'talla' es uno de los indices, si es mayor de '0', hay algun dato
                if (idxc>0)
                {
                        //Si hay exito, guardo los valores de los candidatos...
                        xp=x+1;  //se suma 1 ya que x es columna relativa a la imagen
                        dato.datos[0][0]=xp;
                        dato.datos[0][1]=candidato;
                        dato.datos[0][2]=longitud;
                        dato.datos[0][3]=valor;
                        Incluye(salida,dato,&idxs,idxc);
                        valido=1;
                        if((exito==0)&&(antcol==0)&&(x!=0))
                        {
                                antcol=1;
                                exito=1;
                                x=x-3;
                        }
                        else if((exito==1))
                        {
                                x=x-1;
                                antcol=0;
                        }
                }
        }
        *salida.fil_real=idxs;
        if (idxs>0)
        {
                ClasificaFila(salida,1);
                for(i=0;i<idxs;i++)    //salida es una variable de ncol*4
                {
                        for(j=0;j<4;j++)
                        {
                                salida1.datos[i][j]=salida.datos[i][j];
                        }
                }
        }
        *salida1.fil_real=idxs;
        LiberarMatriz(salida);
        LiberarMatriz(dato);
        LiberarMatriz(brkcode);
        LiberarMatriz(secuencia);
        LiberarMatriz(trozo);
        LiberarMatriz(flt);
        LiberarMatriz(flt2);
        LiberarMatriz(auxiliar);
        LiberarMatriz(picos);
        LiberarMatriz(puntos);
        LiberarMatriz(linea);
        //free(linea.datos[0]);
        }

//---------------------------------------------------------------------------
//      Nombre: Fdeltat
//      Funcion:
```

```
//              Funcion auxiliar, para averiguar si loa periodos de la secuencia dada
//              pueden pertenecer a un codigo Barker, al tener una relacion '2T,2T,T,T,T'.
//      Parametros de entrada:
//              tMatriz secuencia -> trozo con los posibles codigos Barker.
//              int *valet -> indice de validacion.
//              float *distancia -> distancia del codigo Barker.
//              tMatriz brkcode -> matriz con la relacion de correlacion Barker.
//      Parametros de salida:
//              ninguno, ya que son pasados por referencia.
//-----------------------------------------------------------------------------
void Fdeltat(tMatriz secuencia,int *valet,float *distancia,tMatriz brkcode)
{
        float talla,T,offset,maximo,umbralt;
        int ncol,aux,i;

        tMatriz codigo,diferencia;

        TamanoMatriz(&aux,&ncol,secuencia);
        codigo=CrearMatriz(aux,ncol);
        diferencia=CrearMatriz(aux,ncol);
        talla=secuencia.datos[0][5]-secuencia.datos[0][0];
        if (talla>=14)
        {
                T=talla/7;
                offset=secuencia.datos[0][0];
                for(i=0;i<ncol;i++)
                {
                        codigo.datos[0][i]=secuencia.datos[0][i]-offset;
                }
                //construimos la matriz brkcode
                brkcode.datos[0][0]=0;
                brkcode.datos[0][1]=Redondeo(2*T);
                brkcode.datos[0][2]=Redondeo(4*T);
                brkcode.datos[0][3]=Redondeo(5*T);
                brkcode.datos[0][4]=Redondeo(6*T);
                brkcode.datos[0][5]=Redondeo(7*T);

                Resta(diferencia,brkcode,codigo);
                Abs(diferencia);
                maximo=Max(diferencia);
                umbralt=(float)ceil(2*(double)talla/100);
                if (maximo<=umbralt)
                {
                        *valet=1;
                        *distancia=talla;
                }
                else
                {
                        *valet=0;
                        *distancia=0;
                }
        }
        else
        {
                *valet=0;
                *distancia=0;
                brkcode.datos[0][0]=0;
        }
        LiberarMatriz(codigo);
        LiberarMatriz(diferencia);
}
//-----------------------------------------------------------------------------
//      Nombre: Tstrozomx
//      Funcion:
//              Funcion auxiliar, para averiguar si existe un Barker o no
//              en el trozo considerado.
//      Parametros de entrada:
//              tMatriz trozo -> trozo a considerar.
//              float *distancia -> tamaño del posible codigo Barker.
//              tMatriz brkcode -> matriz con los valores de un codigo Barker.
//              int *vale -> codigo de validacion.
//              float *valor -> valor de correlacion.
//      Parametros de salida:
//              Ninguno, ya que son pasados por referencia.
//-----------------------------------------------------------------------------
void Tstrozomx(tMatriz trozo,float *distancia,tMatriz brkcode,int *vale,
                                        float *valor)
{
        float umbral=(float)4/7;          //Umbralizacion del codigo Barker7
        tMatriz barker,Total,result;
        int ncol,aux,i;
        int t1,t2,t3,t4;

        result=CrearMatriz(1,1);
        //Normalizamos a los valores +1,-1
        Normal(trozo,trozo);
```

```
        TamanoMatriz(&aux,&ncol,trozo);
        for(i=0;i<ncol;i++)
        {
                trozo.datos[0][i]=2*trozo.datos[0][i]-1;
        }
        //Recuperacion de los tiempos del codigo, del array 'brkcode':
        t1=brkcode.datos[0][1];
        t2=brkcode.datos[0][2];
        t3=brkcode.datos[0][3];
        t4=brkcode.datos[0][4];

        //Sintesis del codigo Barker
        Total=CrearMatriz(aux,ncol);
        barker=CrearMatriz(1,(int)(*distancia+1));
        UnosMatriz(1,(int)(*distancia+1),barker); //Definicion inicial de la matriz...
        for(i=0;i<t1-1;i++){barker.datos[0][i]=-1;}
        for(i=t2-1;i<t3-1;i++){barker.datos[0][i]=-1;}
        for(i=t4-1;i<(int)*distancia;i++){barker.datos[0][i]=-1;}
        for(i=0;i<ncol;i++)
        {
                Total.datos[0][i]=trozo.datos[0][i]*barker.datos[0][i];
        }

        Sum(Total,result,2);
        *valor=result.datos[0][0];
        //Determinacion de la validez del codigo Barker respecto a su umbral:
        LiberarMatriz(result);
        *valor=*valor/(*distancia+1);

        if (*valor>umbral)
        {
                *vale=1;

        }
        else
        {
                *vale=0;
        }

        LiberarMatriz(barker);
        LiberarMatriz(Total);
}
//-------------------------------------------------------------------------
//      Nombre: Incluye
//      Funcion:
//              Funcion para incluir una nueva fila en una matriz.
//              Parametros de entrada:
//                      tMatriz matriz -> matriz en la cual se quiere incluir una nueva fila.
//                      tMatriz dato -> vector o matriz de datos a introducir.
//                      int *idx -> indice que indica la fila libre en la matriz.
//                      int talla -> indice que indica el numero de vectores de datos que
//                                                              contiene la matriz dato.
//      Parametros de salida:
//              Ninguno, ya que son pasados por referencia.
//-------------------------------------------------------------------------
void Incluye(tMatriz matriz,tMatriz dato,int *idx,int talla)
{
        int j, i, nfil, ncol,aux;
        TamanoMatriz(&nfil,&ncol,dato);
        for(j=0;j<talla;j++)
        {
                aux=*idx;
                for(i=0;i<ncol;i++)
                {
                        matriz.datos[aux][i]=dato.datos[j][i];
                }
                *idx=*idx+1;
        }
}
```

fcirc.c

```
#include "vision.h"

int MejorMarca(tMatriz clusters, tMatriz marcaOK);

//-------------------------------------------------------------------------
//      Nombre: Fcirc.
//      Funcion:
//              Se encarga de distribuir las tareas de para la deteccion y procesado.
//              Realiza llamadas a las tareas busqueda Barker, agrupamiento (cluster)
//              codigos Barker, deteccion de centroides y lectura codigo Barras.
//      Parametros de entrada:
//              tMatriz circulos -> coordenadas [u,v] de los puntos del plano imagen:
//                                                              p1,p2,p3,p4.
```

```
//              tMatriz salida1 -> codigos Barker detectados
//              tMatriz nmarcas -> valor del codigo de Barras leido
//              tMatriz I -> Matriz con los datos de la imagen
//      Parametros de salida:
//              Devuelve un 1 si todo se procesó correctamente y un 0 si hubo errores
//-------------------------------------------------------------------------

int Fcirc(tMatriz circulos,tMatriz salida1,tMatriz nmarcas,tMatriz I)
{
        int k;
        //int num1,k;
        int fmax,cmax;
        //int fmax,cmax,aux;
        tMatriz salida2,marcaOK;
        //tMatriz auxiliar;
        tMatriz C1,C2,C3,C4;

        //Comienzo del algoritmo:busqueda de codigos Barker7

        Fbrk(I,salida1); //'salida1' tiene las coordenadas de todos
                        //los codigos Barker7 detectados

        //Segundo paso, si existen codigos ->agrupamos 'salida1' en clusters:
        TamanoMatriz(&fmax,&cmax,salida1);
        if (fmax&&cmax!=0)
        {
                //El formato de salida2 es: salida2[i][j]={coli,colf,fila_media,long_media}
                salida2=CrearMatriz(10,4); //como maximo 10 posibles marcas por imagen
                marcaOK=CrearMatriz(1,4);// Aquí se almacenará la marca buena, con
                                                // el formato de salida2

                Fclst(salida1,salida2);
                //printf("\nCluster");

                if (!MejorMarca(salida2, marcaOK))
                // Tomamos, de entre todos los cluster de salida2,
                // aquel correspondiente a la marca con el barker + grande
                {
                        LiberarMatriz(salida2);
                        LiberarMatriz(marcaOK);
                        return 0;
                }

                //Tercer paso: para el cluster de marcaOK creamos una subventana
                //y la procesamos para obtener las coordenadas de los 'pi'.

                C1=CrearMatriz(1,6);
                C2=CrearMatriz(1,6);
                C3=CrearMatriz(1,6);
                C4=CrearMatriz(1,6);
                //auxiliar=CrearMatriz(1,4);

                //Tomamos dos subventanas, una a izquierda (WU) y otra a derecha (WR)
                Fsubwin(C1,C2,C3,C4,I,marcaOK);           // Cambio marcaOK por auxiliar, ya
                                                        // que tienen lo mismo y la funcion
                                                        // no modifica su contenido

                        // Chequeo 3 buenos
                if (
                  (C1.datos[0][0] && C1.datos[0][1]) +
                (C2.datos[0][0] && C2.datos[0][1]) +
                (C3.datos[0][0] && C3.datos[0][1]) +
                (C4.datos[0][0] && C4.datos[0][1]) < 3)
                {
                        LiberarMatriz(salida2);
                        LiberarMatriz(marcaOK);
                        LiberarMatriz(C1);
                        LiberarMatriz(C2);
                        LiberarMatriz(C3);
                        LiberarMatriz(C4);
                        return 0;
                }

                //Organizamos la forma de dar los datos de los circulos
                //el formato es: circulos={u1 v1 u2 v2 ... v4      1}

                for(k=0;k<2;k++)
                {
                        circulos.datos[0][k]=C1.datos[0][k];
                        circulos.datos[0][k+2]=C2.datos[0][k];
                        circulos.datos[0][k+4]=C3.datos[0][k];
                        circulos.datos[0][k+6]=C4.datos[0][k];
                }
                circulos.datos[0][8]=0+1;

                k=Barras(marcaOK,I,nmarcas);
                LiberarMatriz(salida2);
```

```
                LiberarMatriz(marcaOK);
                //LiberarMatriz(auxiliar);
                LiberarMatriz(C1);
                LiberarMatriz(C2);
                LiberarMatriz(C3);
                LiberarMatriz(C4);
                return k;
        }
        else
        {
                printf ("\nNo vi nada de nada");
                fflush (stdout);
                return 0;
        }
}

int MejorMarca(tMatriz clusters, tMatriz marcaOK)
// Tomamos, de entre todos los cluster de salida2,
// aquel correspondiente a la marca con el barker + grande
// Devuelve 1 si todo fue bien y 0 si ninguno tiene untamaño suficiente
// para leer el código de barras
{
        int mejor_cluster=-1;
        float tam, max_tam=45;
        int i;

        for (i=0;i<*clusters.fil_real;i++)
        {
                tam=Redondeo(clusters.datos[i][3])+10;
                if (tam > max_tam)
                {
                        max_tam=tam;
                        mejor_cluster=i;
                }
        }

        if ( mejor_cluster != -1 )
        {
                for (i=0;i<4;i++)
                        marcaOK.datos[0][i]=clusters.datos[mejor_cluster][i];
                return 1;
        }
        else return 0;
}
```

fctr.c

```
#include "vision.h"

#define  umbralbn 0.75

void Cfitnew(float *c, tMatriz linea, float offset);

//-------------------------------------------------------------------------
//      Nombre: Fctr.
//      Funcion:
//              Obtiene los centros de los dos circulos presentes en la imagen de la
//              subventana 'W'. Esta imagen procede de la segmentacion, a izquierda
//              y derecha. de la imagen original tras haber encontrado en ella un
//              codigo Barker valido.
//              Cada subimagen 'W' contiene un circulo, ajustado en sus laterales
//              a los bordes de la imagen: solo hay que detectar su psicion en
//              vertical -> solo puede haber 1 circulo.
//              Si en la busqueda de minimos en la segmentacion. aparecieran mas de
//              dos picos, se invalida la busqueda del circulo y se devuelven
//              coordenadas [0 0] en 'centro'.
//      Parametros de entrada:
//              tMatriz Wana -> subventana que contiene el circulo.
//              float umbral -> es el umbral de la segmentacion para buscar los picos
//                                                      umbral=0.3.
//              tMatriz centro -> coordenadas del centro del circulo.
//              float *fili -> fila inicial.
//              float *filf -> fila final.
//      Parametros de salida:
//              Ninguno, ya que son pasados por referencia.
//-------------------------------------------------------------------------
void Fctr(tMatriz WAna,float umbral,tMatriz centro,float *fili,float *filf)
{
        int col,i,j,ndatos,aux,npp,npn,x,a;
        float maxpos1,*media=NULL,picosp1,picosn1,cfil,ccol;
        tMatriz W,W2,auxiliar,auxiliar2,sumfils,auxiliar4;
        tMatriz transit,maxpos,picosp,picosn,bordei,auxpos;
        tMatriz sumfils1,codigo,auxiliar3,sumcols,borded;

        TamanoMatriz(&ndatos,&col,WAna);
```

```
//segmentacion del circulo a partir de la version b/n.
W=CrearMatriz(ndatos,col);
for(i=0;i<ndatos;i++)
{
        for(j=0;j<col;j++)
        {
                W.datos[i][j]=WAna.datos[i][j]>umbralbn;
        }
}
auxiliar=CrearMatriz(ndatos,1);
Sum(W,auxiliar,2);
sumfils=CrearMatriz(ndatos,1);
Normal(auxiliar,sumfils);
codigo=CrearMatriz(ndatos,1);
for(i=0;i<ndatos;i++)
{
        codigo.datos[i][0]=sumfils.datos[i][0]>umbral;
}
auxiliar2=CrearMatriz(ndatos,1);
Diff(codigo,auxiliar2);
transit=CrearMatriz(1,ndatos);
transit.datos[0][0]=0;
for(i=0;i<ndatos-1;i++)
{
        transit.datos[0][i+1]=auxiliar2.datos[i][0];
}

//Busqueda de las franjas de las subventanas
picosp=CrearMatriz(1,ndatos);
picosn=CrearMatriz(1,ndatos);
maxpos=CrearMatriz(ndatos,1);
if (ndatos)
        media=(float *)malloc(ndatos*sizeof(float));
else
        media=(float *)malloc(sizeof(float));
auxpos=CrearMatriz(1,ndatos);
Buscar(transit,picosp,'>',0);
Buscar(transit,picosn,'<',0);
auxiliar4=CrearMatriz(1,ndatos);
Traspuesta(auxiliar4,sumfils);
Buscar(auxiliar4,auxpos,'>',0.95);
LiberarMatriz(auxiliar4);
Traspuesta(maxpos,auxpos);
Media(maxpos,1,media);
maxpos1=Redondeo(*media);
TamanoMatriz(&aux,&npp,picosp);
TamanoMatriz(&aux,&npn,picosn);
if ((npp==0)||(npn==0))
{
        centro.datos[0][0]=0.0;
        centro.datos[1][0]=0.0;  //¡Error!, no se ha encontrado nada.
        *fili=0;
        *filf=0;
}
else
{
        bordei=CrearMatriz(1,npp);
        Buscar(picosp,bordei,'<',maxpos1);
        x=(int)Max(bordei);
        picosp1=picosp.datos[0][x];
        borded=CrearMatriz(1,npn);
        Buscar(picosn,borded,'>',maxpos1);
        x=(int)Min(borded);
        picosn1=picosn.datos[0][x];
        picosp1=picosp1-1;
        if (picosp1==0)
        {
                picosp1=1;
        }
        picosn1=picosn1+1;
        if (picosn1>ndatos)
        {
                picosn1=ndatos; //picosp1 y picosn1 son relativos a la imagen
        }
        *fili=picosp1+1;   //indices absolutos respecto de la imagen
        *filf=picosn1+1;
        sumfils1=CrearMatriz((int)(picosn1-picosp1+1),1);
        Sum(WAna,auxiliar,2);
        j=0;
        for(i=picosp1;i<=picosn1;i++)
        {
                sumfils1.datos[j][0]=auxiliar.datos[i][0];
                j++;

        }
        //centroide de fila
```

```
                Cfitnew(&cfil,sumfils1,picosp1+1);
                //encontrado el circulo, se busca el centroide en analogico
                W2=CrearMatriz((int)(picosn1-picosp1+1),col);
                a=0;
                for(i=picosp1;i<=picosn1;i++)
                {
                        for(j=0;j<col;j++)
                        {
                                W2.datos[a][j]=WAna.datos[i][j];
                        }
                        a++;

                }
                sumcols=CrearMatriz(col,1);
                auxiliar3=CrearMatriz(1,col);
                Sum(W2,auxiliar3,1);
                Traspuesta(sumcols,auxiliar3);
                LiberarMatriz(auxiliar3);
                Cfitnew(&ccol,sumcols,1);
                centro.datos[0][0]=cfil;
                centro.datos[1][0]=ccol;
                LiberarMatriz(borded);
                LiberarMatriz(bordei);
                LiberarMatriz(sumfils1);
                LiberarMatriz(W2);
                LiberarMatriz(sumcols);
        }
        free (media);
        LiberarMatriz(W);
        LiberarMatriz(auxiliar);
        LiberarMatriz(auxiliar2);
        LiberarMatriz(sumfils);
        LiberarMatriz(codigo);
        LiberarMatriz(transit);
        LiberarMatriz(picosp);
        LiberarMatriz(picosn);
        LiberarMatriz(maxpos);
        LiberarMatriz(auxpos);

}
//-------------------------------------------------------------------------
//      Nombre: Cfitnew.
//      Funcion:
//              El metodo de centroides se ve muy afectado por la desigual iluminacion
//    de las lineas, se intenta mejor un ajuste polinomico asimilando la
//              curva de sumas de filas/columnas a una parabola: f(x)=ax^2 + bx + c
//              su centro estara en el maximo, por lo tanto en:
//                                                              centro=-b/(2a)
//      Parametros de entrada:
//              float *c -> centro
//    tMatriz linea -> trozo de linea perteneciente al circulo.
//              float offset -> coeficiente de ajuste.
//      Parametros de salida:
//              Ninguno, ya que son pasados por referencia.
//-------------------------------------------------------------------------
void Cfitnew(float *c, tMatriz linea, float offset)
{
        float p[3];
        int longitud, aux;
        TamanoMatriz(&longitud,&aux,linea);

        //obtencion de los coeficientes y del centro ajustado:
        Polyfit(longitud,linea,p);

        *c=(offset-p[1]/(2*p[2]))-1;
}
```

fsubwin.c

```
#include "vision.h"

#define  umbralbn 0.75

void Cfitnew(float *c, tMatriz linea, float offset);

//-------------------------------------------------------------------------
//      Nombre: Fctr.
//      Funcion:
//              Obtiene los centros de los dos circulos presentes en la imagen de la
//              subventana 'W'. Esta imagen procede de la segmentacion, a izquierda
//              y derecha. de la imagen original tras haber encontrado en ella un
//              codigo Barker valido.
//              Cada subimagen 'W' contiene un circulo, ajustado en sus laterales
//              a los bordes de la imagen: solo hay que detectar su psicion en
```

```
//              vertical -> solo puede haber 1 circulo.
//              Si en la busqueda de minimos en la segmentacion. aparecieran mas de
//              dos picos, se invalida la busqueda del circulo y se devuelven
//              coordenadas [0 0] en 'centro'.
//      Parametros de entrada:
//              tMatriz Wana -> subventana que contiene el circulo.
//              float umbral -> es el umbral de la segmentacion para buscar los picos
//                                                      umbral=0.3.
//              tMatriz centro -> coordenadas del centro del circulo.
//              float *fili -> fila inicial.
//              float *filf -> fila final.
//      Parametros de salida:
//              Ninguno, ya que son pasados por referencia.
//-------------------------------------------------------------------------
void Fctr(tMatriz WAna,float umbral,tMatriz centro,float *fili,float *filf)
{
        int col,i,j,ndatos,aux,npp,npn,x,a;
        float maxpos1,*media=NULL,picosp1,picosn1,cfil,ccol;
        tMatriz W,W2,auxiliar,auxiliar2,sumfils,auxiliar4;
        tMatriz transit,maxpos,picosp,picosn,bordei,auxpos;
        tMatriz sumfils1,codigo,auxiliar3,sumcols,borded;

        TamanoMatriz(&ndatos,&col,WAna);

        //segmentacion del circulo a partir de la version b/n.
        W=CrearMatriz(ndatos,col);
        for(i=0;i<ndatos;i++)
        {
                for(j=0;j<col;j++)
                {
                        W.datos[i][j]=WAna.datos[i][j]>umbralbn;
                }
        }
        auxiliar=CrearMatriz(ndatos,1);
        Sum(W,auxiliar,2);
        sumfils=CrearMatriz(ndatos,1);
        Normal(auxiliar,sumfils);
        codigo=CrearMatriz(ndatos,1);
        for(i=0;i<ndatos;i++)
        {
                codigo.datos[i][0]=sumfils.datos[i][0]>umbral;
        }
        auxiliar2=CrearMatriz(ndatos,1);
        Diff(codigo,auxiliar2);
        transit=CrearMatriz(1,ndatos);
        transit.datos[0][0]=0;
        for(i=0;i<ndatos-1;i++)
        {
                transit.datos[0][i+1]=auxiliar2.datos[i][0];
        }

        //Busqueda de las franjas de las subventanas
        picosp=CrearMatriz(1,ndatos);
        picosn=CrearMatriz(1,ndatos);
        maxpos=CrearMatriz(ndatos,1);
        if (ndatos)
                media=(float *)malloc(ndatos*sizeof(float));
        else
                media=(float *)malloc(sizeof(float));
        auxpos=CrearMatriz(1,ndatos);
        Buscar(transit,picosp,'>',0);
        Buscar(transit,picosn,'<',0);
        auxiliar4=CrearMatriz(1,ndatos);
        Traspuesta(auxiliar4,sumfils);
        Buscar(auxiliar4,auxpos,'>',0.95);
        LiberarMatriz(auxiliar4);
        Traspuesta(maxpos,auxpos);
        Media(maxpos,1,media);
        maxpos1=Redondeo(*media);
        TamanoMatriz(&aux,&npp,picosp);
        TamanoMatriz(&aux,&npn,picosn);
        if ((npp==0)||(npn==0))
        {
                centro.datos[0][0]=0.0;
                centro.datos[1][0]=0.0;  //¡Error!, no se ha encontrado nada.
                *fili=0;
                *filf=0;
        }
        else
        {
                bordei=CrearMatriz(1,npp);
                Buscar(picosp,bordei,'<',maxpos1);
                x=(int)Max(bordei);
                picosp1=picosp.datos[0][x];
                borded=CrearMatriz(1,npn);
                Buscar(picosn,borded,'>',maxpos1);
                x=(int)Min(borded);
```

```
                picosn1=picosn.datos[0][x];
                picosp1=picosp1-1;
                if (picosp1==0)
                {
                        picosp1=1;
                }
                picosn1=picosn1+1;
                if (picosn1>ndatos)
                {
                        picosn1=ndatos; //picosp1 y picosn1 son relativos a la imagen
                }
                *fili=picosp1+1;   //indices absolutos respecto de la imagen
                *filf=picosn1+1;
                sumfils1=CrearMatriz((int)(picosn1-picosp1+1),1);
                Sum(WAna,auxiliar,2);
                j=0;
                for(i=picosp1;i<=picosn1;i++)
                {
                        sumfils1.datos[j][0]=auxiliar.datos[i][0];
                        j++;

                }
                //centroide de fila

                Cfitnew(&cfil,sumfils1,picosp1+1);
                //encontrado el circulo, se busca el centroide en analogico
                W2=CrearMatriz((int)(picosn1-picosp1+1),col);
                a=0;
                for(i=picosp1;i<=picosn1;i++)
                {
                        for(j=0;j<col;j++)
                        {
                                W2.datos[a][j]=WAna.datos[i][j];
                        }
                        a++;

                }
                sumcols=CrearMatriz(col,1);
                auxiliar3=CrearMatriz(1,col);
                Sum(W2,auxiliar3,1);
                Traspuesta(sumcols,auxiliar3);
                LiberarMatriz(auxiliar3);
                Cfitnew(&ccol,sumcols,1);
                centro.datos[0][0]=cfil;
                centro.datos[1][0]=ccol;
                LiberarMatriz(borded);
                LiberarMatriz(bordei);
                LiberarMatriz(sumfils1);
                LiberarMatriz(W2);
                LiberarMatriz(sumcols);
        }
        free (media);
        LiberarMatriz(W);
        LiberarMatriz(auxiliar);
        LiberarMatriz(auxiliar2);
        LiberarMatriz(sumfils);
        LiberarMatriz(codigo);
        LiberarMatriz(transit);
        LiberarMatriz(picosp);
        LiberarMatriz(picosn);
        LiberarMatriz(maxpos);
        LiberarMatriz(auxpos);

}
//-------------------------------------------------------------------------
//      Nombre: Cfitnew.
//      Funcion:
//              El metodo de centroides se ve muy afectado por la desigual iluminacion
//    de las lineas, se intenta mejor un ajuste polinomico asimilando la
//              curva de sumas de filas/columnas a una parabola: f(x)=ax^2 + bx + c
//              su centro estara en el maximo, por lo tanto en:
//                                                              centro=-b/(2a)
//      Parametros de entrada:
//              float *c -> centro
//    tMatriz linea -> trozo de linea perteneciente al circulo.
//              float offset -> coeficiente de ajuste.
//      Parametros de salida:
//              Ninguno, ya que son pasados por referencia.
//-------------------------------------------------------------------------
void Cfitnew(float *c, tMatriz linea, float offset)
{
        float p[3];
        int longitud, aux;
        TamanoMatriz(&longitud,&aux,linea);

        //obtencion de los coeficientes y del centro ajustado:
        Polyfit(longitud,linea,p);
```

```
        *c=(offset-p[1]/(2*p[2]))-1;
}
```

posicion.c

```
#include "vision.h"

//-------------------------------------------------------------------------
//      Nombre: Posicion.
//      Funcion:
//              Algoritmo de posicionamiento.
//              Se toman como entrda los puntos (u,v) y los angulos (alfa,beta).
//              Una vex halladas las distancias z1 y z3 (a los puntos p1 y p3),
//              determinan el resto de coordenadas (xi,yi) y de ahi los valores
//              de gamma. Una ves determinada R, se recupera todo el vector
//              C=(T,Omega).
//      Parametros de entrada:
//              tMatriz puntos -> coordenadas [u,v] de los puntos del plano imagen:
//                                                                  p1,p2,p3,p4.
//              float pantilt[] -> angulos del pan_tilt [alfa,beta]
//              float camara[] -> parametros de la camara.
//              float marca[] -> datos de la marca.
//              float *angulos -> angulos alfa,beta,gamma.
//              float *POS -> matriz de posiciones [x,y,z] recuperadas.
//      Parametros de salida:
//              Ninguno ya que son pasados por referencia.
//-------------------------------------------------------------------------

void Posicion(tMatriz puntos,float pantilt[],float camara[],float marca[],
                                        float*angulos, float *POS)
{
        tMatriz A,B,R,At,Ainv,R1,R2,z1,z3;
        tMatriz D,D1,D2,E,E1,E2,Gamma,Gamma1,Gamma2,pt1,pt3;
        float u1,v1,u2,v2,u3,v3,u4,v4,alfa,beta,gamma,k;
        float deltaH,deltaV,lambda,a12,b12,c12,x1,y1,x3,y3;
        float gms,gmc,r11,r12,r13,r21,r22,r23,r31,r32,r33;

        //Creacion de las matrices necesarias
        z1=CrearMatriz(1,1);
        z3=CrearMatriz(1,1);
        D=CrearMatriz(4,1);
        D1=CrearMatriz(2,1);
        D2=CrearMatriz(2,1);
        E=CrearMatriz(4,2);
        E1=CrearMatriz(2,2);
        E2=CrearMatriz(2,2);
        Gamma=CrearMatriz(2,1);
        Gamma1=CrearMatriz(2,1);
        Gamma2=CrearMatriz(2,1);
        pt1=CrearMatriz(1,3);
        pt3=CrearMatriz(1,3);

        //Recuperacion de las varibles intermedias
        alfa=pantilt[0];
        beta=pantilt[1];
        u3=puntos.datos[0][7];   //punto 3
        v3=puntos.datos[0][6];
        u4=puntos.datos[0][5];   //punto 4
        v4=puntos.datos[0][4];
        u1=puntos.datos[0][3];   //punto 1
        v1=puntos.datos[0][2];
        u2=puntos.datos[0][1];  //punto 2
        v2=puntos.datos[0][0];

        if(u1==0&&u2==0&&u3==0&&u4==0&&v1==0&&v2==0&&v3==0&&v4==0)
        {
                angulos[0]=0;angulos[1]=0;angulos[2]=0;
                POS[0]=0;POS[1]=0;POS[2]=0;
        }
        else
        {
                A=CrearMatriz(2,1);
                B=CrearMatriz(2,1);
                At=CrearMatriz(1,2);
                R1=CrearMatriz(1,1);
                R2=CrearMatriz(1,2);
                Ainv=CrearMatriz(1,1);
                //Cosenos directores a partir de la matriz de rotacion R
                a12=-sin(beta);
                b12=sin(alfa)*cos(beta);
                c12=-cos(alfa)*cos(beta);

                //Longitudes de simbolo y otras constantes
                deltaH=2*marca[0];       //Longitud horizontal -> distancia horizontal
```

```
deltaV=2*marca[1];       //Anchura vertical -> distancia vertical
lambda=camara[0];

//Obtencion de z1:
//Creacion de la matriz A
A.datos[0][0]=u2-u1;
A.datos[1][0]=v2-v1;
//Creacion de la matriz B
B.datos[0][0]=deltaV*(lambda*a12-u2*c12);
B.datos[1][0]=deltaV*(lambda*b12-v2*c12);
//Computo de la profundidad 'z'
Traspuesta(At,A);
Multiplica(R1,At,A);     //z1=(deltaV*(lambda*b12-v2*c12))/(v2-v1);
Inversa(R1,Ainv);
Multiplica(R2,Ainv,At);
Multiplica(z1,R2,B);
//Obtencion de z3:
//Creacion de la matriz A
A.datos[0][0]=u4-u3;
A.datos[1][0]=v4-v3;
//Creacion de la matriz B
B.datos[0][0]=-deltaV*(lambda*a12-u4*c12);
B.datos[1][0]=-deltaV*(lambda*b12-v4*c12);
//Computo de la profundidad 'z'
Traspuesta(At,A);
Multiplica(R1,At,A);     //z1=(-deltaV*(lambda*b12-v4*c12))/(v4-v3);
Inversa(R1,Ainv);
Multiplica(R2,Ainv,At);
Multiplica(z3,R2,B);
//Liberamos memoria para poder reutilizar las variables
LiberarMatriz(A);
LiberarMatriz(At);
LiberarMatriz(Ainv);
LiberarMatriz(R1);
LiberarMatriz(R2);
LiberarMatriz(B);

//Computo de 'x' e 'y'
x1=(z1.datos[0][0]*u1)/lambda;
y1=(z1.datos[0][0]*v1)/lambda;
x3=(z3.datos[0][0]*u3)/lambda;
y3=(z3.datos[0][0]*v3)/lambda;

//Determinacion del angulo 'gamma'
//Constante 'k'
k=1/deltaH;

//Matriz 'D'
D1.datos[0][0]=(u4-u1)*z1.datos[0][0];
D1.datos[1][0]=(v4-v1)*z1.datos[0][0];
D2.datos[0][0]=(u2-u3)*z3.datos[0][0];
D2.datos[1][0]=(v2-v3)*z3.datos[0][0];

//Matriz 'E'
E1.datos[0][0]=lambda;
E1.datos[0][1]=u4*sin(alfa);
E1.datos[1][0]=0;
E1.datos[1][1]=(lambda*cos(alfa)+v4*sin(alfa));
E2.datos[0][0]=-lambda;
E2.datos[0][1]=-u2*sin(alfa);
E2.datos[1][0]=0;
E2.datos[1][1]=-(lambda*cos(alfa)+v2*sin(alfa));

//Computo del angulo 'gamma' inicial:

At=CrearMatriz(2,2);
R1=CrearMatriz(2,2);
R2=CrearMatriz(2,2);
Ainv=CrearMatriz(2,2);

Traspuesta(At,E1);
Multiplica(R1,At,E1);
Inversa(R1,Ainv);
Multiplica(R2,Ainv,At);
Multiplica(Gamma1,R2,D1);
Gamma1.datos[0][0]=Gamma1.datos[0][0]*k;
Gamma1.datos[1][0]=Gamma1.datos[1][0]*k;

Traspuesta(At,E2);
Multiplica(R1,At,E2);
Inversa(R1,Ainv);
Multiplica(R2,Ainv,At);
Multiplica(Gamma2,R2,D2);
Gamma2.datos[0][0]=Gamma2.datos[0][0]*k;
Gamma2.datos[1][0]=Gamma2.datos[1][0]*k;

D.datos[0][0]=D1.datos[0][0];
```

```
D.datos[1][0]=D1.datos[1][0];
D.datos[2][0]=D2.datos[0][0];
D.datos[3][0]=D2.datos[1][0];

E.datos[0][0]=E1.datos[0][0];
E.datos[0][1]=E1.datos[0][1];
E.datos[1][0]=E1.datos[1][0];
E.datos[1][1]=E1.datos[1][1];
E.datos[2][0]=E2.datos[0][0];
E.datos[2][1]=E2.datos[0][1];
E.datos[3][0]=E2.datos[1][0];
E.datos[3][1]=E2.datos[1][1];

//Liberamos memoria para poder reutilizar las variables
LiberarMatriz(At);
LiberarMatriz(Ainv);
LiberarMatriz(R1);
LiberarMatriz(R2);

At=CrearMatriz(2,4);
R1=CrearMatriz(2,2);
R2=CrearMatriz(2,4);
Ainv=CrearMatriz(2,2);

Traspuesta(At,E);
Multiplica(R1,At,E);
Inversa(R1,Ainv);
Multiplica(R2,Ainv,At);
Multiplica(Gamma,R2,D);
Gamma.datos[0][0]=Gamma.datos[0][0]*k;
Gamma.datos[1][0]=Gamma.datos[1][0]*k;

//Liberamos memoria para poder reutilizar las variables
LiberarMatriz(At);
LiberarMatriz(Ainv);
LiberarMatriz(R1);
LiberarMatriz(R2);

if (Signo(Gamma1.datos[1][0])!=Signo(Gamma2.datos[1][0]))
{
        Gamma.datos[1][0]=0;
        //En caso de incertidumbre, iniciamos a cero 'cos(gamma)'
}

gms=Gamma.datos[0][0];
if (gms>1)
{
        gms=1;
}
else if (gms<-1)
{
        gms=-1;
}
gmc=Gamma.datos[1][0];
if (gmc>1)
{
        gmc=1;
}
else if (gmc<-1)
{
        gmc=-1;
}

//Los angulos solo pueden estar entre 'pi' y 2*pi'
if (gmc<0)
{
        gms=PI+asin(gms);
}
else
{
        gms=2*PI-asin(gms);
}
gamma=gms;

//Determinacion de la matriz Rotacion R:
r31=a12;
r32=b12;
r33=c12;
r21=cos(beta)*sin(gamma);
r22=sin(alfa)*sin(beta)*sin(gamma)+cos(alfa)*cos(gamma);
r23=cos(alfa)*sin(beta)*sin(gamma)-sin(alfa)*cos(gamma);
r11=cos(beta)*cos(gamma);
r12=sin(alfa)*sin(beta)*cos(gamma)-cos(alfa)*sin(gamma);
r13=cos(alfa)*sin(beta)*cos(gamma)+sin(alfa)*sin(gamma);
R=CrearMatriz(3,3);
R.datos[0][0]=r11;R.datos[0][1]=r12;R.datos[0][2]=r13;
R.datos[1][0]=r21;R.datos[1][1]=r22;R.datos[1][2]=r23;
```

```
                R.datos[2][0]=r31;R.datos[2][1]=r32;R.datos[2][2]=r33;

                //Obtencion del vector de posicion POS
                //coordenadas de pt1 recuperadas en el SCC
                pt1.datos[0][0]=-x1/2;pt1.datos[0][1]=-y1/2;pt1.datos[0][2]=-z1.datos[0][0]/2;
                //coordenadas de pt3 recuperadas en el SCC
                pt3.datos[0][0]=-x3/2;pt3.datos[0][1]=-y3/2;pt3.datos[0][2]=-z3.datos[0][0]/2;
                Ainv=CrearMatriz(3,3);
                R1=CrearMatriz(1,3);
                R2=CrearMatriz(1,3);
                Inversa(R,Ainv);
                Suma(R1,pt1,pt3);
                Multiplica(R2,R1,Ainv);
                POS[0]=R2.datos[0][0];
                POS[1]=R2.datos[0][1];
                POS[2]=R2.datos[0][2];

                angulos[0]=alfa;
                angulos[1]=beta;
                angulos[2]=gamma;
                LiberarMatriz(R);
                LiberarMatriz(Ainv);
                LiberarMatriz(R1);
                LiberarMatriz(R2);
        }
        //Liberamos toda la memoria usada
        LiberarMatriz(pt1);
        LiberarMatriz(pt3);
        LiberarMatriz(z1);
        LiberarMatriz(z3);
        LiberarMatriz(D);
        LiberarMatriz(D1);
        LiberarMatriz(D2);
        LiberarMatriz(E);
        LiberarMatriz(E1);
        LiberarMatriz(E2);
        LiberarMatriz(Gamma);
        LiberarMatriz(Gamma1);
        LiberarMatriz(Gamma2);
}
```

proceso.c

```
#include "vision.h"
#include "colas.h"
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include <string.h>
#include <time.h>

//Datos de la camara:
#define lambda                  1               //Distancia focal normalizada
#define pixel                   8.4e-6          //pixeles cuadrados
#define SensorXMax              (5.4e-3)/2      //tamaño maximo horizontal
#define SensorYMax              (4e-3)/2        //tamaño maximo vertical
#define sigma2                  0.02
//Datos de la marca
#define whor                    0.215/2
#define wver                    0.13/2
#define ESCALA                  5       // Para obtener x,y en la escala del mapa. (cada cm son 5m)
#define MAX_LONG_NODO           200
#define MAX_NODOS_MAPA          200

extern int colaid;
extern FILE *id_fmapa;          // Abierto en captura.c
extern char letra_mapa;

int PosicionReal (tMatriz total, struct buf_msg_vi *paquete);
int Calcula_jerarquia(int nodo_visto);

//-------------------------------------------------------------------------
//              Nombre: Proceso
//      Funcion:
//                      Gestionar las tareas de deteccion y procesado
//              Parametros de entrada:
//                      tMatriz Imagen -> Matriz que contiene los datos de la imagen a procesar
//              Parametros de salida:
//                      Devuelve 1 si todo fue bien, 0 si hubo errores y hay que tomar otra imagen
//-------------------------------------------------------------------------
int Proceso(tMatriz Imagen)
{
        float camara[5],marca[2],pantilt[2],angulos[3],POS[3];
        tMatriz circulos,salida,salidacalib,nmarcas;
        tMatriz puntos,total;
```

```
int i;
int j;
float X,Y,R,Fi;

struct buf_msg_vi paquete;
struct timespec tiempoV;

camara[0]=lambda;
camara[1]=pixel*153.8;
camara[2]=SensorXMax;
camara[3]=SensorYMax;
camara[4]=sigma2;

marca[0]=whor;
marca[1]=wver;
        // CIRCULOS es de 1 fila y 9 columnas, ya que sólo procesamos una marca
circulos=CrearMatriz(1,9);
nmarcas=CrearMatriz(1,9);

salida=CrearMatriz(200,4);               //reservamos suficiente espacio para almacenar
                                         //las posibles columnas con codigos Barker
salidacalib=CrearMatriz(1,9);

if (!Fcirc(circulos,salida,nmarcas,Imagen))
//Llamada a la tarea de deteccion. Si casca, volvemos al proceso principal y repetimos
{
        LiberarMatriz(circulos);
        LiberarMatriz(nmarcas);
        LiberarMatriz(salida);
        LiberarMatriz(salidacalib);
        return 0;
}

Calibra(circulos,salidacalib);    //Calibracion de los centroides

pantilt[0]=PI/2;
pantilt[1]=0;
puntos=CrearMatriz(1,9);
total=CrearMatriz(1,9);

memcpy (&puntos.datos[0][0],&salidacalib.datos[0][0],9*sizeof(float));

Posicion(puntos,pantilt,camara,marca,angulos,POS);     //Obtencion de los angulos
X=POS[0];Y=POS[1];
R=sqrt(pow(X,2)+pow(Y,2));
Fi=(angulos[2]-3*PI/2)*180/PI;    //Fi=gamma-3pi/2
//total={nºmarca marca X Y alfa beta gamma R Fi }
total.datos[0][0]=0+1;
total.datos[0][1]=nmarcas.datos[0][1];
total.datos[0][2]=X;
total.datos[0][3]=Y;
total.datos[0][4]=angulos[0];
total.datos[0][5]=angulos[1];
total.datos[0][6]=angulos[2];
total.datos[0][7]=R;
total.datos[0][8]=Fi;
 //La variable total contiene todos los parametros buscados

if (!PosicionReal (total,&paquete))
// Calculamos la posicion real y la guardamos en un paquete
{
        LiberarMatriz(total);
        LiberarMatriz(puntos);
        LiberarMatriz(circulos);
        LiberarMatriz(salida);
        LiberarMatriz(salidacalib);
        LiberarMatriz(nmarcas);
        return 0; // Si falla la determinacion de la posicion real repetimos todo
}

// Enviamos el paquete al modulo de control mediante un mensaje
paquete.mtype=MSG_VISION;// Paquete vision


i=msgsnd(colaid,(struct msgbuf *)&paquete, sizeof(paquete)-sizeof(long), IPC_NOWAIT);

if (i==-1)
        perror ("\nVision: Error al enviar mensaje");

LiberarMatriz(total);
LiberarMatriz(puntos);
LiberarMatriz(circulos);
LiberarMatriz(salida);
LiberarMatriz(salidacalib);
LiberarMatriz(nmarcas);

return 1;
```

```
}

int PosicionReal (tMatriz total, struct buf_msg_vi *paquete)
{
        // En la matriz total=[Nº,MARCA,X,Y,alfa,beta,gamma,R,Fi]
        // Hay que coger del mapa la posicion de la MARCA, y luego implementar
        // la transformación de la ecuación 4.28, pag.41, del trabajo 2 de Marta

        float posMarca[3];

        char Nodo[MAX_LONG_NODO];
        int j;

        static int jerarq_ant=4;
                // 3 -> Sala (pasillo lat)
                // 2 -> Pasillo central
                // 1 -> Planta

        int nodo_visto, jerarq_act;
        static int nodo_ant=0, signo_ant=0;
        int signo;

        // Recuperamos la posición de la marca

        rewind(id_fmapa);
        j=0;
        nodo_visto=total.datos[0][1];

        /* buscando el nodo cuyo campo nombre coincida con el deseado */
        do     /* mientras no encuentre un nodo cuyo nombre coincida*/
        {
                fgets(Nodo,MAX_LONG_NODO,id_fmapa);

        }while((nodo_visto!=atoi(Nodo))&&((j++)<=MAX_NODOS_MAPA));


        /* En la variable "nodo" tengo el nodo cuyo nombre coincide con nodo_Partida */
        if(j<MAX_NODOS_MAPA)     // Encontrado
        {
                        // Obtengo nodo del mapa
                strcpy (paquete->nodo, strtok (Nodo," "));// Nombre del nodo
                paquete->nodo[3]='-';
                paquete->nodo[4]=letra_mapa;
                paquete->nodo[5]='\0';

                        // Obtengo coordenadas de la marca del mapa
                posMarca[0]=((float)atoi(strtok('\0'," ")))/100;   //Obtengo el valor del x del nodo
                posMarca[1]=((float)atoi(strtok('\0'," ")))/100;   //Obtengo el valor del y del nodo
                posMarca[2]=(float)atoi(strtok('\0'," "));
                        //Obtengo la orientacion de la normal en grados

                        // Calculo la jerarquia del nodo visto para coger el ángulo leido
                        // o el opuesto, en función de la jerarquia del último nodo visto
                jerarq_act=Calcula_jerarquia(nodo_visto);

                if (jerarq_act<jerarq_ant)
                        signo=-180;
                else // si es mayor o igual esel mismo angulo
                        signo=0;

                if (jerarq_ant==4 && jerarq_act<=2)
                        signo=0;         // Caso especial: planta y pasillo central(la 1a vez)

                if (nodo_visto==nopdo_ant)
                        signo=signo_ant;

                nodo_ant=nodo_visto;
                signo_ant=signo;

                jerarq_ant=jerarq_act;

                        // Transformación de coordenadas
                        // Los datos sacados del mapa(marca) están en m/5(escala) y grados
                        //Los de total : gamma está en radianes, X e Y en metros

                posMarca[2]+=signo;//angulo marca en dir 'hacia mi'

                // Calculamos la coord. X (en m/5)
                paquete->datos[0]=((total.datos[0][2]*cos(posMarca[2]*2*PI/360))-
                                   (total.datos[0][3]*sin(posMarca[2]*2*PI/360)))/ESCALA;
                paquete->datos[0]+=posMarca[0];
                        // Calculamos la coord. Y (en m/5)
                paquete->datos[1]=((total.datos[0][2]*sin(posMarca[2]*2*PI/360))+
                                   (total.datos[0][3]*cos(posMarca[2]*2*PI/360)))/ESCALA;
                paquete->datos[1]+=posMarca[1];

                        // Calculamos la coord. theta (en grados) -> Orientacion del movil
```

```
                paquete->datos[2]=(posMarca[2]-180)+((total.datos[0][6]*180/PI)-270);
                                //angulo marca en dir 'contra mi' mas (gamma-270)

                if (fabs(paquete->datos[2])>=360)
                        paquete->datos[2]=fmod(paquete->datos[2],360);

                if (paquete->datos[2]<0)
                        paquete->datos[2]+=360;

                return 1;
        }
        else
        {
                printf("\nNodo no encontrado (%d)",(int)total.datos[0][1]);
                return 0;
        }
}

int Calcula_jerarquia(int nodo_visto)
{
        int aver;

        aver=nodo_visto%100;
        if (aver>9)
        {
                aver=aver%10;
                if (aver)
                        return 3;       // habitacion (pe 349,348...)
                else
                        return 2;       // salida pasillo lateral a central (pe 310,320,330..)
        }
        else
                if (aver)
                        return 2;       // habitacion del pasillo central (pe 301 a 309)
                else
                        return 1;       // planta (000,100,200,300)

}
```

ekf.h

```
void init_ekf ();
void ekf (float v, float omega, float tiempo, Posicion *Xanterior,
          int hay_vision, Posicion Zanterior);
void ekf_prediccion (float V, float omega, float Ts, tMatriz *Xant);
void ekf_correccion ( tMatriz *XPRE, tMatriz Zant);
void fin_ekf ();

#define Radio            0.16
#define Dist             0.52
#define NUMESTADOS       3
#define NUMENTRADAS      2
#define NUMSALIDAS       3
#define CTE1             Ts*Radio/2
#define CTE2             Ts*Radio/Dist
#define ESCALA           5               // Para obtener x,y en la escala del mapa. (cada cm son 5m)

#define RVALOR           0.01
#define QVALOR           0.0001
```

ekf.c

```
#include "libmatrices.h"
#include "gestor.h"
#include "ekf.h"

        /*********************************
        **      E       K       F       **
        *********************************/

tMatriz P;              // Global para el ekf

void init_ekf ()
{
        P=CrearMatriz(3,3);
        CerosMatriz(3,3,P);      // Partimos de datos de vision-> P a cero

        return;
}

void fin_ekf ()
{
        LiberarMatriz(P);
}


void ekf (float v, float omega, float tiempo,
          Posicion *Xanterior, int hay_vision, Posicion Zanterior)
{

        tMatriz X;
        tMatriz Zant;

        X=CrearMatriz(1,NUMESTADOS);
                X.datos[0][0]=Xanterior->x;
                X.datos[0][1]=Xanterior->y;
                X.datos[0][2]=(Xanterior->Theta)*PI/180;  // Paso a radianes

        ekf_prediccion (v, omega, tiempo, &X);
                // Ahora, en X está XPRE y, en P, P_AST
        if (hay_vision==1)
        {
                Zant=CrearMatriz(1,NUMESTADOS);
                // Mapeo lo que venga en Zanterior a Zant;
                        Zant.datos[0][0]=Zanterior.x;
                        Zant.datos[0][1]=Zanterior.y;
                        Zant.datos[0][2]=Zanterior.Theta*PI/180;  // Paso a radianes

                ekf_correccion ( &X, Zant);
                LiberarMatriz(Zant);
        }

        //copio resultados de la matriz a la estructura a devolver por referencia.
        //Ahora, en Xanterior va la nueva posición
                Xanterior->x=X.datos[0][0];
                Xanterior->y=X.datos[0][1];

                // Vuelvo a grados
                X.datos[0][2]=X.datos[0][2]*180/PI;

                        // Ya hecho en cada funcion por separado

                Xanterior->Theta=X.datos[0][2];
```

```
                LiberarMatriz(X);

                return;          // Se devuelve Xanterior
}

                /*********************************
                 ***     Etapa de Predicción      ***
                 *********************************/

void ekf_prediccion (float V, float omega, float Ts, tMatriz *Xant)
                // Ts será el parámetro tiempo de ekf()
                //      Resultados: XPRE y P_AST se devuelven por referencia en Xant y Pant
{
                // Definición de variables

        tMatriz aux1,aux2,aux3,aux4;      // Para resultados intermedios, 3x3.

        tMatriz G; // Matriz de estados del modelo de
                   // estimación (odométrico) linealizado, 3x3.

        tMatriz W;  // Relaciona el ruido de medida de estado
                    //(odometria) con el vector de estado

        int i;
        tMatriz Q;       // Matriz 2x2 ruido

        fflush (stdout);

        /* Inicializo las estructuras tMatriz */

        aux1=CrearMatriz(3,3);
        aux2=CrearMatriz(3,3);
        aux3=CrearMatriz(3,3);
        aux4=CrearMatriz(3,3);
        Q=CrearMatriz(2,2);
        CreaIdentidad(Q);
        G=CrearMatriz(3,3);
        CreaIdentidad(G);        // No es la identidad, cambio elementos despues.
        W=CrearMatriz(3,2);

        // Elementos de Q
        for (i=0;i<2;i++)
                Q.datos[i][i]=QVALOR;

        /* Aqui voy a definir los elementos de G y W que varian en cada iteración */
        G.datos[2][0] = -Ts*V*sin(Xant->datos[0][2]);
        G.datos[2][1] = Ts*V*cos(Xant->datos[0][2]);
        W.datos[0][0] = CTE1*cos(Xant->datos[0][2]);
        W.datos[0][1] = CTE1*sin(Xant->datos[0][2]);
        W.datos[1][0] = CTE2;
        W.datos[1][1] = CTE1*cos(Xant->datos[0][2]);
        W.datos[2][0] = CTE1*sin(Xant->datos[0][2]);
        W.datos[2][1] = -CTE2;

                /* Obtengo la matriz de estados predicho XPRE */
        Xant->datos[0][0]+= Ts*V*cos(Xant->datos[0][2])/ESCALA;
        Xant->datos[0][1]+= Ts*V*sin(Xant->datos[0][2])/ESCALA;
                // omega en rad/seg.
        Xant->datos[0][2]+= Ts*omega;

        if (fabs(Xant->datos[0][2])>=2*PI)
                Xant->datos[0][2]=fmod (Xant->datos[0][2]*180/PI,360)*PI/180;

        if (Xant->datos[0][2]<0)
                Xant->datos[0][2]+=2*PI;

                // En Xant queda guardado el valor de XPRE

                /* Hallando Pk+1estimadak, la matriz de innovacion */

        // Obtengo Pk+1estimadak=P_AST=G*PK*G'.....
        Traspuesta(aux1,G);              // Traspuesta (resultado,original);
        Multiplica(aux2,G,P);            // Multiplica (resultado,A,B)
        Multiplica(aux3,aux2,aux1);

        // .... + GM*Q*GM'
        Multiplica(aux1,W,Q);
        Traspuesta(aux2,W);
        Multiplica(aux4,aux1,aux2);
        Suma(P,aux3,aux4);               // Suma (resultado,A,B)
                // En Pant queda guardado el valor de P_AST

        LiberarMatriz(aux1);
        LiberarMatriz(aux2);
        LiberarMatriz(aux3);
        LiberarMatriz(aux4);
        LiberarMatriz(Q);
```

```
        LiberarMatriz(G);
        LiberarMatriz(W);

        // Fin de ekf_prediccion()
        // Los resultados son XPRE y P_AST
}


                /*********************************
                 ***     Etapa de Corrección      ***
                 *********************************/
void ekf_correccion ( tMatriz *XPRE, tMatriz Zant)
{
                 // Definición de variables

        int i;
        tMatriz aux1,aux2,aux3;          // Para resultados intermedios, 3x3.
        tMatriz C;                       // Matriz del modelo de estados (odométrico)
        tMatriz K;                       // Ganancia de Kalman (Etapa de corrección), 3x3.
        tMatriz IKC;                     // Para resultado intermedio en el cálculo de PK1,3x3.
        tMatriz R;                       // Matriz 3x3 ruido
        tMatriz Yout;
        tMatriz Pout;

                /* Inicializo las estructuras tMatriz */
        aux1=CrearMatriz(3,3);
        aux2=CrearMatriz(3,3);
        aux3=CrearMatriz(3,3);
        C=CrearMatriz(3,3);
        CreaIdentidad(C);        // Realmente C es la matriz identidad.
        K=CrearMatriz(3,3);
        C=CrearMatriz(3,3);
        CreaIdentidad(C);        // Realmente C es la matriz identidad.
        K=CrearMatriz(3,3);
        IKC=CrearMatriz(3,3);
        R=CrearMatriz(3,3);
        CreaIdentidad(R);
        Yout=CrearMatriz(1,NUMESTADOS);
        Pout=CrearMatriz(3,3);


                // Defino los elementos de R
        for (i=0;i<3;i++)
                R.datos[i][i]=RVALOR;

                /* Hallando K */

                // Obtengo la matriz C*P_AST*C'
        Multiplica(aux1,C,P);
        Traspuesta(aux2,C);
        Multiplica(aux3,aux1,aux2);

        // Se suma ....+R
        Suma(aux1,aux3,R);

        //.... y se invierte....
        Inversa(aux1,aux2);              // Inversa (origen,resultado)

        // Finalmente K=P_AST*C'*....
        Traspuesta(aux1,C);
        Multiplica(aux3,P,aux1);
        Multiplica(K,aux3,aux2);


                /* Hallando PK1 */

        // Obtengo la matriz I-K*C
        CreaIdentidad(aux1);
        Multiplica(aux2,K,C);
        Resta(IKC,aux1,aux2);            // Resta (resultado,A,B)

        // Obtengo PK1=IKC*P_AST
        Multiplica(Pout,IKC,P);

                /* Hallando XK1 estimada */

        // Obtengo XK1= [I-KC]*.... + K*ZK
        Resta(aux1,Zant,*XPRE);

                // Ajustamos la diferencia de orientación
        if (aux1.datos[0][2]>=PI)
                aux1.datos[0][2]-=2*PI;

        if (aux1.datos[0][2]<PI)
                aux1.datos[0][2]+=2*PI;

        Traspuesta(aux3,aux1);
```

```
        Multiplica(aux2,K,aux3);
        Traspuesta(aux3,aux2);

        Suma(Yout,*XPRE,aux3);

        // Fin de ekf_correccion()
        // Los resultados son Yout y Pout, los pasamos a XPRE y P_AST
        Multiplica (*XPRE, Yout, C);      // C es la matriz identidad
        if (fabs(XPRE->datos[0][2])>=2*PI)
                XPRE->datos[0][2]=fmod (XPRE->datos[0][2]*180/PI,360)*PI/180;

        if (XPRE->datos[0][2]<0)
                XPRE->datos[0][2]+=2*PI;

        Multiplica (P, Pout, C);

        LiberarMatriz(aux1);
        LiberarMatriz(aux2);
        LiberarMatriz(aux3);
        LiberarMatriz(C);
        LiberarMatriz(K);
        LiberarMatriz(R);
        LiberarMatriz(IKC);
        LiberarMatriz(Yout);
        LiberarMatriz(Pout);

}
```

libmatrices.h

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>

#define PI 3.141592

typedef struct
{
        int fil,col;                    //nº de filas y columnas asignadas a la matriz
        int *fil_real, *col_real;//nº real de filas y columnas, por defecto igual a fil y col.
        float **datos;                  //array de datos
}tMatriz;

tMatriz CrearMatriz(int N_FILAS,int N_COLUMNAS);
void LiberarMatriz(tMatriz matriz);
void TamanoMatriz(int *N_Fila, int *N_Col, tMatriz matriz);
void CerosMatriz(int filas, int columnas, tMatriz matriz);
void UnosMatriz(int filas, int columnas, tMatriz matriz);
void CreaIdentidad(tMatriz matriz);
void Sum(tMatriz x, tMatriz y, int indicador);
void ClasificaFila (tMatriz x, int n);
void Buscar(tMatriz entrada, tMatriz salida, int indicador, float valref);
void Abs(tMatriz matriz);
float Redondeo(float valor);
float Max(tMatriz matriz);
float Min(tMatriz matriz);
void Diff(tMatriz entrada, tMatriz salida);
void Media(tMatriz entrada, int indicador, float* result);
float MVS(float vect[],int numelem);
float Signo(float valor);
void Normal(tMatriz entrada,tMatriz salida);
void Polyfit(int X, tMatriz Y, float *v);

void Multiplica(tMatriz result, tMatriz A, tMatriz B);
void Suma(tMatriz result, tMatriz A, tMatriz B);
void Resta(tMatriz result, tMatriz A, tMatriz B);
void Traspuesta(tMatriz traspuesta, tMatriz A);
void Inversa(tMatriz A, tMatriz Ainv);
```

libmatrices.c

```
#include "libmatrices.h"

tMatriz CrearMatriz(int N_Fil,int N_Col)
{
        // Quizas sería interesante no hacer asignación dinámica de memoria

        /*declaracion de variables*/
        tMatriz matriz;
        int i;
```

```
        /*reserva de memoria para la matriz*/

        if ((matriz.datos=(float **)malloc((N_Fil)*sizeof(float *)))==NULL)
        {
                printf("Error al asignar memoria\n");
                exit(-1);
        }
        for(i=0;i<N_Fil;i++)
        {
                if ((matriz.datos[i]=(float *)malloc((N_Col)*sizeof(float)))==NULL)
                {
                        printf("Error al asignar memoria\n");
                        exit(-1);
                }
        }
        matriz.fil=N_Fil;
        matriz.col=N_Col;

        matriz.fil_real=(int *)malloc(1*sizeof(int));
        matriz.col_real=(int *)malloc(1*sizeof(int));
        *matriz.fil_real=N_Fil;  //Valores por defecto
        *matriz.col_real=N_Col;

        return(matriz);
}

void LiberarMatriz(tMatriz matriz)
{

        int i;

        //Liberar la memoria asignada a cada una de las filas
        for(i=0;i<matriz.fil;i++)
        {
                free(matriz.datos[i]);
        }
        //Liberar la memoria asignada
        free(matriz.datos);

        free(matriz.fil_real);
        free(matriz.col_real);
}

void TamanoMatriz(int *N_Fila, int *N_Col, tMatriz matriz)
{
        *N_Fila=*matriz.fil_real;
        *N_Col=*matriz.col_real;
}

void CerosMatriz(int filas, int columnas, tMatriz matriz)
{
        int i,j;
        for(i=0;i<filas;i++)
        {
                for(j=0;j<columnas;j++)
                {
                        matriz.datos[i][j]=0;
                }
        }
}

void UnosMatriz(int filas, int columnas, tMatriz matriz)
{
        int i,j;
        for(i=0;i<filas;i++)
        {
                for(j=0;j<columnas;j++)
                {
                        matriz.datos[i][j]=1;
                }
        }
}

void CreaIdentidad(tMatriz matriz)
{
        int i;

        CerosMatriz (*matriz.fil_real, *matriz.col_real, matriz);
        for(i=0;i<*matriz.fil_real;i++)
        {
                matriz.datos[i][i]=1;
        }
}

void Sum(tMatriz x, tMatriz y, int indicador)
{
        int i,j, ncol,nfil;
```

```
        float suma;
        TamanoMatriz(&nfil,&ncol,x);
        switch(indicador)
        {
                case 1:
                        for(i=0;i<ncol;i++)
                        {
                                suma=0;
                                for (j=0;j<nfil;j++)
                                {
                                        suma=suma+x.datos[j][i];
                                }
                                y.datos[0][i]=suma;
                        }
                        *y.fil_real=1;
                        *y.col_real=ncol;
                break;
                case 2:
                        for(i=0;i<nfil;i++)
                        {
                                suma=0;
                                for(j=0;j<ncol;j++)
                                {
                                        suma=suma+x.datos[i][j];
                                }
                                y.datos[i][0]=suma;
                        }
                        *y.fil_real=nfil;
                        *y.col_real=1;
                break;
        }
}

void ClasificaFila (tMatriz x, int n)
{

        int n_filas,n_cols;
        int i,j,s;
        tMatriz aux;
        TamanoMatriz(&n_filas,&n_cols,x);

        aux=CrearMatriz(n_filas,n_cols);

        s=1;
        while (s==1)
        {
                s=0;
                for (i=1;i<n_filas;i++)
                {
                        if (x.datos[i-1][n-1]>x.datos[i][n-1])
                        {
                                s=1;
                                for(j=0;j<n_cols;j++)
                                {
                                        aux.datos[i-1][j]=x.datos[i][j];
                                        aux.datos[i][j]=x.datos[i-1][j];
                                        x.datos[i][j]=aux.datos[i][j];
                                        x.datos[i-1][j]=aux.datos[i-1][j];
                                }
                        }
                }
        }
        LiberarMatriz(aux);
}

void Buscar(tMatriz entrada, tMatriz salida, int indicador, float valref)
{
        int ncol,aux,x,i;
        TamanoMatriz(&aux,&ncol,entrada);
        switch (indicador)
        {
                case '!':

                        x=0;
                        for(i=0;i<ncol;i++)
                        {
                                if (entrada.datos[0][i]!=valref)
                                {
                                        salida.datos[0][x]=(float)i;
                                        x++;
                                }
                        }
                        break;
                case '>':
                        x=0;
                        for(i=0;i<ncol;i++)
                        {
```

```
                                if (entrada.datos[0][i]>valref)
                                {
                                        salida.datos[0][x]=(float)i;
                                        x++;
                                }
                        }
                        break;

                case '<':
                        x=0;
                        for(i=0;i<ncol;i++)
                        {
                                if (entrada.datos[0][i]<valref)
                                {
                                        salida.datos[0][x]=(float)i;
                                        x++;
                                }
                        }
                        break;
                case '=':
                        x=0;
                        for(i=0;i<ncol;i++)
                        {
                                if (entrada.datos[0][i]==valref)
                                {
                                        salida.datos[0][x]=(float)i;
                                        x++;
                                }
                        }
                        break;
        }
        *salida.col_real=x;
}

void Abs(tMatriz matriz)
{
        int nfil,ncol;
        int i,j;
        TamanoMatriz(&nfil,&ncol,matriz);
        for(i=0;i<nfil;i++)
        {
                for(j=0;j<ncol;j++)
                {
                        if (matriz.datos[i][j]<0)
                        {
                                matriz.datos[i][j]=matriz.datos[i][j]*-1;
                        }
                }
        }
}

float Redondeo(float valor)
{
        double aux;
        float referencia, resultado;
        aux=floor((double)valor);
        referencia=valor-(float)aux;
        if (referencia>=0.5)
        {
                resultado=(float)aux+1;
        }
        else
        {
                resultado=(float)aux;
        }
        return(resultado);
}

float Max(tMatriz matriz)
{
        float res;
        int i,j,nfil,ncol;

        TamanoMatriz(&nfil,&ncol,matriz);
        res=matriz.datos[0][0];
   if(nfil||ncol==0)
        {
                res=0;
        }
        for(i=0;i<nfil;i++)
        {
                for(j=0;j<ncol;j++)
                {
                        if(matriz.datos[i][j]>res)
                        {
                                res=matriz.datos[i][j];
                        }
```

```
                }
        }
        return(res);
}

float Min(tMatriz matriz)
{
        float res;
        int i,j,nfil,ncol;

        TamanoMatriz(&nfil,&ncol,matriz);

        res=matriz.datos[0][0];
        for(i=0;i<nfil;i++)
        {
                for(j=0;j<ncol;j++)
                {
                        if(matriz.datos[i][j]<res)
                        {
                                res=matriz.datos[i][j];
                        }
                }
        }
        return(res);
}

void Diff(tMatriz entrada, tMatriz salida)
{
        int i,j, nfil, ncol;
        TamanoMatriz(&nfil,&ncol,entrada);
        for(i=0;i<nfil-1;i++)
        {
                for(j=0;j<ncol;j++)
                {
                        salida.datos[i][j]=entrada.datos[i+1][j]-entrada.datos[i][j];
                }
        }
        *salida.fil_real=nfil-1;
}

void Media(tMatriz entrada, int indicador, float* result)
{
        int nfil,ncol,i,j;
        float suma;
        //float *result;

        TamanoMatriz(&nfil,&ncol,entrada);

        switch (indicador)
        {
                case 1:
                        for(j=0;j<ncol;j++)
                        {
                                suma=0;
                                for(i=0;i<nfil;i++)
                                {
                                        suma=suma+entrada.datos[i][j];
                                }
                                result[j]=suma/(float)nfil;
                        }
                break;
                case 2:
                        for(i=0;i<nfil;i++)
                        {
                                suma=0;
                                for(j=0;j<ncol;j++)
                                {
                                        suma=suma+entrada.datos[i][j];
                                }
                                result[i]=suma/(float)ncol;
                        }
                break;
         }
}

float MVS(float vect[],int numelem)
{
        int i;
        float suma,resultado,*vector;
        suma=0;
        vector=(float *)malloc(numelem*sizeof(float));

        for(i=0;i<numelem;i++)
        {
                vector[i]=vect[i];       //salvaguardamos el contenido del vector
                if (vector[i]<0){vector[i]=vector[i]*-1;}
                vector[i]=(float)pow((double)vector[i],2);
```

```
                suma=suma+vector[i];
        }
        resultado=(float)pow((double)suma,0.5);
        free(vector);
        return(resultado);

}

float Signo(float valor)
{
        float resul;
        if (valor>0)
        {
                resul=1;
        }
        else if (valor<0)
        {
                resul=-1;
        }
        else if (valor==0)
        {
                resul=0;
        }
        return(resul);
}

void Normal(tMatriz entrada,tMatriz salida)
{
        float a,b;
        int i,j,nfil,ncol;
        a=Max(entrada);
        b=Min(entrada);
        TamanoMatriz(&nfil,&ncol,entrada);
        for(i=0;i<nfil;i++)
        {
                for(j=0;j<ncol;j++)
                {
                        salida.datos[i][j]=((entrada.datos[i][j]-b)/(a-b));
                }
        }
}

void Polyfit(int X, tMatriz Y, float *v)
{
        tMatriz A,At,Aux,Aux2,Aux3,Aux4;
        int i;


        A=CrearMatriz(X,3);
        At=CrearMatriz(3,X);
        Aux=CrearMatriz(3,3);
        Aux2=CrearMatriz(3,3);
        Aux3=CrearMatriz(3,3);
        Aux4=CrearMatriz(3,3);

        //formamos matriz A

        for(i=1;i<=X;i++)
        {
                A.datos[i-1][0]=1;
                A.datos[i-1][1]=i;
                A.datos[i-1][2]=i*i;
        }
        Traspuesta(At,A);
        Multiplica(Aux,At,A);
        Inversa(Aux,Aux2);
        Multiplica(Aux3,At,Y);
        Multiplica(Aux4,Aux2,Aux3);

        v[0]=Aux4.datos[0][0];
        v[1]=Aux4.datos[1][0];
        v[2]=Aux4.datos[2][0];

        LiberarMatriz(A);
        LiberarMatriz(At);
        LiberarMatriz(Aux);
        LiberarMatriz(Aux2);
        LiberarMatriz(Aux3);
        LiberarMatriz(Aux4);

}

void Multiplica( tMatriz result, tMatriz A, tMatriz B)
{

        int i,j,k;
        int nfil_a,ncol_a,nfil_b,ncol_b;
```

```
        TamanoMatriz(&nfil_a,&ncol_a,A);
        TamanoMatriz(&nfil_b,&ncol_b,B);

        if(ncol_a!=nfil_b)
        {
                printf ("\nNo es posible la multiplicacion: cols_a=%d y fils_b=%d\n",ncol_a,nfil_b);
                exit(-1);       /* No es posible la multiplicacion*/
        }

        *result.fil_real=nfil_a;
        *result.col_real=ncol_b;

        for(i=0;i<nfil_a;i++)
        {
                for(k=0;k<ncol_b;k++)
                {
                        result.datos[i][k]=0;
                        for(j=0;j<ncol_a;j++)
                        {

        result.datos[i][k]=result.datos[i][k]+(A.datos[i][j]*B.datos[j][k]);
                        }
                }
        }
}

void Suma(tMatriz result, tMatriz A, tMatriz B)
{

        int i,j;
        int nfil_a,ncol_a,nfil_b,ncol_b;

        TamanoMatriz(&nfil_a,&ncol_a,A);
        TamanoMatriz(&nfil_b,&ncol_b,B);

        if(ncol_a!=ncol_b||nfil_a!=nfil_b)       /* No es posible la suma*/
        {
                printf ("\nNo es posible la suma: cols_a=%d y cols_b=%d\t;\tfils_a=%d
                        y fils_b=%d\n",ncol_a,ncol_b,nfil_a,nfil_b);
                exit(-1);               }
        else
        {
                *result.fil_real=nfil_a;
                *result.col_real=ncol_a;
        }

        for(i=0;i<nfil_a;i++)
        {
                for(j=0;j<ncol_a;j++)
                {

                        result.datos[i][j]=A.datos[i][j]+B.datos[i][j];
                }
        }
}

void Resta(tMatriz result, tMatriz A, tMatriz B)
{

        int i,j;
        int nfil_a,ncol_a,nfil_b,ncol_b;

        TamanoMatriz(&nfil_a,&ncol_a,A);
        TamanoMatriz(&nfil_b,&ncol_b,B);

        if(ncol_a!=ncol_b||nfil_a!=nfil_b)       /* No es posible la resta*/
        {
                printf ("\nNo es posible la resta: cols_a=%d y cols_b=%d\t;
                        \tfils_a=%d y fils_b=%d\n",ncol_a,ncol_b,nfil_a,nfil_b);
                exit(-1);
        }
        else
        {
                *result.fil_real=nfil_a;
                *result.col_real=ncol_a;
        }

        for(i=0;i<nfil_a;i++)
        {
                for(j=0;j<ncol_a;j++)
                {
                        result.datos[i][j]=A.datos[i][j]-B.datos[i][j];
                }
        }
}
```

```
void Traspuesta(tMatriz traspuesta,tMatriz A)
{

        int i,j;
        int nfil,ncol;
        TamanoMatriz(&nfil,&ncol,A);
        for(i=0;i<ncol;i++)
        {
                for(j=0;j<nfil;j++)
                {
                        traspuesta.datos[i][j]=A.datos[j][i];
                }
        }
        *traspuesta.fil_real=ncol;
        *traspuesta.col_real=nfil;
}

void Inversa(tMatriz A, tMatriz Ainv)
{
        int i,j,k,l,fila,columna,col,fil,N;
        float factor;

        tMatriz Aux,I,Aux2;
        TamanoMatriz(&fila,&columna,A);

        if(fila!=columna) exit(-1);       // No es posible hallar la inversa

        N=fila;
        //Reservamos memoria
        Aux=CrearMatriz(N,N);
        Aux2=CrearMatriz(N,N);
        I=CrearMatriz(N,N);

        for(i=0;i<N;i++)
        {
                for(j=0;j<N;j++)
                {
                        Aux.datos[i][j]=0.0;Aux2.datos[i][j]=0.0;
                        if(i==j)//creamos matriz unitaria
                        {
                                I.datos[i][j]=1.0;
                        }
                        else
                        {
                                I.datos[i][j]=0.0;
                        }
                }
        }

        for(k=0;k<N;k++)
        {
                if(A.datos[0][0]==0)     //Si el elemento pivote es cero intercambiamos fila.
                {
                        for(j=0;j<N;j++)
                        {
                                Aux.datos[j][0]=A.datos[j][0];
                                A.datos[j][0]=A.datos[j][1];
                                A.datos[j][1]=Aux.datos[j][0];

                                Aux2.datos[j][0]=I.datos[j][0];
                                I.datos[j][0]=I.datos[j][1];
                                I.datos[j][1]=Aux2.datos[j][0];
                        }
                }

                for(l=k+1;l<N;l++)
                {
                        if(A.datos[k][k]!=0.0)
                        {
                                factor=A.datos[k][l] / A.datos[k][k];
                                for(j=0;j<N;j++)
                                {
                                        A.datos[j][l]=A.datos[j][l]-factor*A.datos[j][k];
                                        I.datos[j][l]=I.datos[j][l]-factor*I.datos[j][k];
                                }
                        }
                }

        }

        for(k=N-1;k>=0;k--)
        {
                for(l=k-1;l>=0;l--)
                {
                        factor=A.datos[k][l]/ A.datos[k][k];
                        for(j=0;j<N;j++)
```

```
                        {
                                A.datos[j][l]=A.datos[j][l]-factor*A.datos[j][k];
                                I.datos[j][l]=I.datos[j][l]-factor*I.datos[j][k];
                        }
                }
        }



        for(col=0;col<N;col++)
        {
                for(fil=0;fil<N;fil++)
                {
                        I.datos[col][fil]=I.datos[col][fil]/ A.datos[fil][fil];
                        Ainv.datos[col][fil]=I.datos[col][fil];
                }
        }

        //Liberamos la memoria asignada
        LiberarMatriz(Aux);
        LiberarMatriz(Aux2);
        LiberarMatriz(I);
}
```

neuron.h

```
/* flags */
#define NEURON_SYNC 0x01
#define NEURON_ERR  0x02
#define NEURON_BUSY 0x04
#define NEURON_EXIST 0x08
#define NEURON_OPENNED 0x10
#define NEURON_NONBLOCK  0x11

/* IOCTLs */
#define NEURONSYNC       0x01    // Ordena una resincronizacion
#define NEURONGFLAGS 0x10
#define NEURONSFLAGS 0x11
#define NEURONGSLAVEMODE 0x18
#define NEURONSSLAVEMODE 0x20


/* neuron Chip parallel port commands */

#define NEURON_CMD_XFER 0x01
#define NEURON_CMD_NULL 0x00
#define NEURON_CMD_RESYNC 0x5A
#define NEURON_CMD_ACKSYNC 0x07
#define NEURON_EOM 0x00

/* NEURON CHIP PARALLEL PORT LINES */

#define NEURON_RW 0x08           /* control register, active low, pin 17 on PC */
#define NEURON_CS 0x02           /* control register, active low, pin 14 on PC */
#define NEURON_HS_A 0x08 /* status register */
#define NEURON_HS_B 0x01 /* data register */

#define NEURON_CS1RW0 NEURON_RW
#define NEURON_CS0RW0 NEURON_RW | NEURON_CS
#define NEURON_CS1RW1 0
#define NEURON_CS0RW1 NEURON_CS



/* Definition of slave modes */
#define NEURON_MODE_A 1
#define NEURON_MODE_B 2
#define NEURON_MODE_NONE 3

#define NEURON_WATCHDOG (HZ*84)/100 /* 0.84sec on a 10MHz NeuronChip */
#define NEURON_BUFFER_SIZE 255

//Driver nuevo
#define BASE_ADDR 0x378
#define STATUS_REG BASE_ADDR + 1
#define CONTROL_REG BASE_ADDR + 2
#define WAIT_HS_DEF 10000

#define DIR_WR 0
#define PARPORT_CONTROL_RW  0x08  /*PARPORT_CONTROL_?, pin 17*/
#define PARPORT_CONTROL_CS  0x02  /* PARPORT_CONTROL_?, pin 14 */
#define PARPORT_STATUS_HS  0x08   /* PARPORT_STATUS_?, pin 15 */

#define CS1_RW1_in 0x20 /* Modo lectura(data_reverse) + ~PARPORT_CONTROL_CS & ~PARPORT_CONTROL_RW */
```

```
#define CS0_RW1 0x22 /* Modo lectura(data_reverse) +  PARPORT_CONTROL_CS | ~PARPORT_CONTROL_RW */
#define CS0_RW0 0x0A /* Modo escritura(data_forward)PARPORT_CONTROL_CS | PARPORT_CONTROL_RW */
#define CS1_RW0 0x08 /* Modo escritura(data_forward)~PARPORT_CONTROL_CS | PARPORT_CONTROL_RW */
#define CS1_RW1_out 0x00

struct neuron_struct
{
        struct pardevice *dev;
        unsigned char sl_mode;
        char *buffer;
        ssize_t data_size;
        unsigned int flags;

        unsigned long jiff_init;

        unsigned char sync_tries;

        struct semaphore port_mutex;
};
```

neuron.c

```
#include <linux/module.h>
#include <linux/init.h>

#include <linux/version.h>
#include <linux/config.h>
#include <linux/errno.h>
#include <linux/kernel.h>
#include <linux/major.h>
#include <linux/sched.h>
#include <linux/devfs_fs_kernel.h>
#include <linux/slab.h>
#include <linux/fcntl.h>
#include <linux/delay.h>

#include <asm/irq.h>
#include <asm/uaccess.h>
#include <asm/system.h>

#include <linux/parport.h>
#include <linux/parport_pc.h>

#include "neuron.h"

#define NEURON_MAJOR 0
#define MAX_NEURON 4
#define MAX_SYNC_TRIES 100


MODULE_DESCRIPTION ("Parallel Port to NeuronChip interface driver");
MODULE_LICENSE ("GPL");
MODULE_AUTHOR ("Jesus Nuevo Chiquero");

static unsigned long base = 0x378;
unsigned long short_base = 0;

static char *slave_mode[MAX_NEURON] = { NULL,  };
MODULE_PARM_DESC(slave_mode, "Slave Mode (A or B) configuration of the NeuronChip. Default: A");
MODULE_PARM(slave_mode,"1-" __MODULE_STRING(MAX_NEURON) "s");

static devfs_handle_t devfs_handle = NULL;
static int neuron_major = NEURON_MAJOR;

struct neuron_struct neuron_table[MAX_NEURON];
unsigned int neuron_count=0;

static int neuron_reset(int minor);

static int neuron_write_byte(int minor, unsigned char d)
{
        int retval=0, wait_hs=WAIT_HS_DEF;
        unsigned long address=BASE_ADDR;

        outb(CS1_RW0,CONTROL_REG);
        wmb();

        outb(d, address);
        wmb();
        while(((inb(STATUS_REG)) & PARPORT_STATUS_HS))
        {
                if(wait_hs--<0)
                {
                        printk(KERN_ERR "Driver Neuron: timeout HS\n");
                        outb(CS1_RW0,CONTROL_REG);
```

```
                        wmb();
                        return -ETIMEDOUT;
                }
        }
        outb(CS0_RW0,CONTROL_REG);
        wmb();
        udelay(1);
        outb(CS1_RW0,CONTROL_REG);
        wmb();

        outb(CS1_RW0, CONTROL_REG);
        return retval;
}

static int neuron_read_byte(int minor, unsigned char *d)
{
        int retval=0, wait_hs=WAIT_HS_DEF;
        unsigned long address=BASE_ADDR;
        char leido;

        outb(CS1_RW1_in, CONTROL_REG);
        wmb();
        udelay(10);

        while((~(inb(CONTROL_REG)) & 0x20))
      {
                if(wait_hs--<0)
            {
                printk(KERN_ERR "Driver Neuron: timeout reverse (read_byte)\n");
                 outb(CS1_RW1_out, CONTROL_REG);
                 wmb();
                 return -ETIMEDOUT;
            }
        }

        while(((inb(STATUS_REG)) & PARPORT_STATUS_HS))
        {
                if(wait_hs--<0)
                {
                        printk(KERN_ERR "Driver Neuron: timeout HS\n");
                        outb(CS1_RW0, CONTROL_REG);
                        wmb();
                        return -ETIMEDOUT;
                }

        }
        leido = inb(address);
        *d=leido;

        printk(KERN_ERR "Driver Neuron (read byte): leido %d\n",(int)leido);
        rmb();
        outb(CS0_RW1,CONTROL_REG);
        wmb();
        udelay(1);
        outb(CS1_RW1_in, CONTROL_REG);
        wmb();

        outb(CS1_RW1_out,CONTROL_REG);

        return retval;
}

static int neuron_release(int minor)
{

        parport_release(neuron_table[minor].dev);
        neuron_table[minor].flags &= ~NEURON_EXIST;
        return 0;
}

static int neuron_sync(int minor)
{
        char command=NEURON_CMD_RESYNC;
        int a;

                if(neuron_write_byte(minor, command)!=0)
                        return -ETIMEDOUT;
                command=0x00;
                if(neuron_write_byte(minor, command)!=0)
                        return -ETIMEDOUT;

        for (a=0;a<MAX_SYNC_TRIES;a++)
        {
        /* lets try to get the ACKSYNC */
                udelay(5);
                neuron_read_byte(minor, &command);
                printk(KERN_ERR "Driver Neuron: Lei %d del neuron\n", command);
```

```
                if(command==NEURON_CMD_ACKSYNC)
                {       /* great! */
                        neuron_table[minor].flags|=NEURON_SYNC;
                        printk(KERN_ERR "Driver Neuron: neuron%d sincronizado\n ", minor);
                        neuron_read_byte(minor, &command); //leo el byte EOM
                        udelay(5);
                        return 0;
                }
        }

        udelay(5);
        return -1;
}

static int neuron_reset(int minor)
{
        neuron_table[minor].flags |= NEURON_BUSY;
        neuron_table[minor].jiff_init = jiffies;

        return( neuron_sync(minor));
}

static int neuron_preempt(void *handle)
{
        return 1;
}

static ssize_t neuron_read(struct file * file, char * buf,
                           size_t count, loff_t *ppos)
{

        unsigned char d;
        unsigned char data_size, bytes_left;
        int a;

        char *kbuf;

        unsigned int minor = MINOR(file->f_dentry->d_inode->i_rdev);

        if(!(neuron_table[minor].flags & NEURON_SYNC))
        {
                neuron_reset(minor);
                return 0;
        }

        for (a=0;a<MAX_SYNC_TRIES;a++)
        {
                neuron_read_byte(minor, &d);      // Aqui leemos el comando del neuron
                if (d!=0)
                        break;
                udelay (10);
        }

        printk (KERN_ERR "Driver Neuron: Lectura, lei comando %d\n",d);
        udelay(5);

                // Comprobamos si nos hemos pasado de tiempo y hay que resetear
        if(jiffies - neuron_table[minor].jiff_init > NEURON_WATCHDOG )
        {
                printk(KERN_ERR "Neuron Chip watchdog");
                udelay(5);
                neuron_reset(minor);
                neuron_table[minor].sync_tries++;
                return 0;
        }

                // Aqui entramos si no tenemos a 1 el flag de sincronismo
                // No estamos sincronizados

        if(d==NEURON_CMD_XFER && (neuron_table[minor].flags & NEURON_OPENNED))
        {
                // ¡¡¡por fin leo!!!
                neuron_read_byte(minor, &data_size);// Leo el tamaño de los datos q va a enviar
                neuron_table[minor].data_size=data_size;
                bytes_left=data_size;
                printk ( KERN_ERR "Estoy leyendo!\n");

                while(bytes_left)                // Aqui se leen los datos
                {
                        neuron_read_byte(minor,&d);
                        neuron_table[minor].buffer[data_size-bytes_left]=d;
                        bytes_left--;
                }
                neuron_table[minor].flags &= ~(NEURON_BUSY);       // Quitamos el flag de ocupado

                kbuf=neuron_table[minor].buffer;
                if(count > neuron_table[minor].data_size)
```

```
                        count = neuron_table[minor].data_size;

                copy_to_user(buf, kbuf, count);   // Devolvemos lo leido
                return count;

        }

        if(d==NEURON_CMD_NULL||d==0x07)
        {
                /* don't use this command in Slave A mode, it doesn't work
                   properly */
                printk ( KERN_ERR "Driver Neuron: Recibi CMD_NULL o 0x07 en lectura\n");
                neuron_table[minor].flags &= ~(NEURON_BUSY);
                return 0;
        }


        // Fuera de sincronismo

        printk ( KERN_ERR "Driver Neuron: Recibi comando desconocido (%d) en lectura\n",d);
        neuron_table[minor].flags &= ~(NEURON_BUSY);
        neuron_table[minor].flags &= ~(NEURON_SYNC);
        neuron_reset(minor);

        return 0;
}

static ssize_t neuron_write(struct file * file, const char * buf,
                            size_t count, loff_t *ppos)
{

        unsigned int minor = MINOR(file->f_dentry->d_inode->i_rdev);
        ssize_t written;

        if(!(neuron_table[minor].flags & NEURON_SYNC))
                return -EIO;

        if(count > NEURON_BUFFER_SIZE)
                count = NEURON_BUFFER_SIZE;

        printk (KERN_ERR "Driver Neuron: Escribiendo\n");

        for(written=0; written < count; written++)
                if(neuron_write_byte (minor, buf[written]))
                        return -ETIMEDOUT;

        neuron_table[minor].flags |= NEURON_BUSY;

        neuron_table[minor].jiff_init = jiffies;
        return written;
}

static int neuron_open(struct inode * inode, struct file * file)
{

        unsigned int minor = MINOR(inode->i_rdev);

        if((parport_claim(neuron_table[minor].dev)))
                return -EBUSY;

        neuron_table[minor].buffer = kmalloc(NEURON_BUFFER_SIZE, GFP_KERNEL);

        if (!(neuron_table[minor].flags && NEURON_SYNC))
                if(neuron_reset(minor))
                        return -ETIMEDOUT;

        neuron_table[minor].flags |= NEURON_OPENNED;

        return 0;
}

static int neuron_ioctl(struct inode *inode, struct file *file,
                        unsigned int cmd, unsigned long arg)
{

        unsigned int minor = MINOR(inode->i_rdev);

        if (minor >= MAX_NEURON)
                return -ENODEV;
        if (!(neuron_table[minor].flags & NEURON_OPENNED))
                return -ENODEV;

        switch ( cmd )
        {
                case NEURONSYNC:
                        neuron_reset (minor);
                        break;
```

```
                case NEURONGFLAGS:
                        if (copy_to_user((char *) arg, &neuron_table[minor].flags,
                                         sizeof(char)))
                                return -EFAULT;
                        break;

                case NEURONSFLAGS:
                        neuron_table[minor].flags=arg;
                        return 0;
                        break;

                case NEURONGSLAVEMODE:
                        if (copy_to_user( (char *)arg, &neuron_table[minor].sl_mode,
                                  sizeof(char)))
                        return -EFAULT;
                        break;

                case NEURONSSLAVEMODE:
                        if ((arg == NEURON_MODE_A) || (arg == NEURON_MODE_B))
                                neuron_table[minor].sl_mode = arg;
                        else
                                return -EFAULT;
                        break;

                default:
                        return -EINVAL;
        }
        return 0;
}

static struct file_operations neuron_fops = {
        owner:          THIS_MODULE,
        write:          neuron_write,
        ioctl:          neuron_ioctl,
        open:           neuron_open,
        read:           neuron_read,
};

static void neuron_attach(struct parport *port)
{
        char name[8];

        if(neuron_table[neuron_count].sl_mode == NEURON_MODE_NONE)
        {
                neuron_count++;
                return ;
        }

        neuron_table[neuron_count].dev = parport_register_device(port,"neuron",neuron_preempt,
                                                                 NULL,NULL,0,
                                                                 (void
*)&neuron_table[neuron_count]);

        if(neuron_table[neuron_count].dev == NULL)
                return;

        neuron_table[neuron_count].flags |= NEURON_EXIST;

        sprintf (name, "%d", neuron_count);
        devfs_register (devfs_handle, name,
                        DEVFS_FL_AUTO_DEVNUM, 0, 0,
                        S_IFCHR | S_IRUGO | S_IWUGO,
                        &neuron_fops, NULL);

        printk(KERN_INFO "neuron%d: using %s.\n", neuron_count, port->name);

        neuron_count++;
        return;
}

static void neuron_detach (struct parport *port)
{

        printk(KERN_INFO "parport detached\n");
        udelay(5);

}

static struct parport_driver neuron_driver =
{
        "neuron",
        neuron_attach,
        neuron_detach,
        NULL
};
```

```
int __init neuron_init(int n)
{
        int i;

        for(i = 0; i < n; i++)
        {
                neuron_table[i].dev=NULL;
                neuron_table[i].flags=0;
                neuron_table[i].jiff_init=0;
                neuron_table[i].sync_tries=0;
                neuron_table[i].buffer=NULL;
        }

        i=devfs_register_chrdev (neuron_major, "neuron", &neuron_fops);

        if(neuron_major == NEURON_MAJOR)   neuron_major = i; /* dinamic */

        devfs_mk_dir (NULL, "neuron", NULL);

        if (parport_register_driver (&neuron_driver))
        {
                printk ("Driver Neuron: unable to register with parport\n");
                return -EIO;
        }

        if (!neuron_count)
                printk (KERN_INFO "Driver Neuron: driver loaded but no devices found\n");

        return 0;
}

static int __init  neuron_init_module(void)
{
        int n=0;

        if(slave_mode[0])
        {
                if (!strncmp(slave_mode[0], "auto", 4))
                {
                        for ( n=0; n < MAX_NEURON; n++)
                                neuron_table[n].sl_mode = NEURON_MODE_A;
                }
                else
                {
                        for (n=0; n< MAX_NEURON && slave_mode[n]; n++)
                        {
                if(!strncmp(slave_mode[n], "none", 4))
                {
                                        neuron_table[n].sl_mode = NEURON_MODE_NONE;
                                        printk(KERN_ERR "Driver Neuron: parport%d
                                                         NEURON_MODE_NONE\n", n);
                }
                else if(!strncmp(slave_mode[n], "A", 1) | !(strncmp(slave_mode[n], "a", 1)))
                {
                                neuron_table[n].sl_mode = NEURON_MODE_A;
                                printk(KERN_ERR "Driver Neuron: parport%d NEURON_MODE_A\n", n);
                }
                else if(!strncmp(slave_mode[n], "B", 1) | !(strncmp(slave_mode[n], "b", 1)))
                {
                                neuron_table[n].sl_mode = NEURON_MODE_B;
                                printk(KERN_ERR "Driver Neuron: parport%d NEURON_MODE_Bn", n);
                }
                else
                        {
                        printk(KERN_ERR "Driver Neuron: bad port specifier `%s'\n", slave_mode[n]);
                        return -ENODEV;
                        }
                }
        }

        }
        else printk(KERN_ERR "Driver Neuron: no mode specified\n");

        return neuron_init(n);
}

static void neuron_cleanup_module(void)
{

        int offset;

        parport_unregister_driver(&neuron_driver);
        for (offset = 0; offset < MAX_NEURON; offset++)
        {
                if(neuron_table[offset].dev == NULL)
                        continue;
                parport_unregister_device(neuron_table[offset].dev);
```

```
        }

        devfs_unregister(devfs_handle);
        devfs_unregister_chrdev(neuron_major, "neuron");

}

module_init(neuron_init_module);
module_exit(neuron_cleanup_module);

EXPORT_NO_SYMBOLS;
```

VI. Presupuesto

En esta sección se realiza una estimación del importe de la ejecución del proyecto. Para ello, se presenta un estudio dividido en diversos apartados en los que se agruparán los gastos según su origen.

VI.1. Coste de los materiales.

En este apartado se engloba el coste del uso de los diversos equipos que han sido necesarios para desarrollar el presente trabajo, describiendo tanto los costes de la parte *hardware* como los de la parte *software*. Por último, se hará un pequeño resumen del conjunto de material de oficina empleado durante la realización del proyecto.

- Recursos *hardware*:

CONCEPTO	Coste hora	Total horas
1. Silla de ruedas equipada con tarjetas de control de bajo nivel	3 €.	100 horas
2. Equipo de desarrollo de redes LonWorks (LonBuider)	3 €.	100 horas
3. Tarjeta interface LonWorks-EPP	1'50 €	100 horas
4. Cámara CCD Sony XC-73CE	1 €	50 horas
5. Tarjeta de video Avermedia	30 €cents.	50 horas

Coste total de los recursos hardware ..... 815 €

- Recursos *software*:

CONCEPTO	Coste hora	Total horas
1. Ordenador PENTIUM III 1000Mhz (con Linux).	1'5 €	1000 horas
2. Impresora HP Laserjet 2100	2 €.	300 horas

Coste total de los recursos software ..... 2100 €

- Material de oficina:

CONCEPTO	Coste
1. Material fungible (papel, repuestos impresora, ...)	60 €.
2. Material no fungible	30 €.

Coste total del material de oficina ..... 90 €

Llegados a este punto, se puede realizar el cálculo final del coste que suponen el conjunto de todos los materiales:

Coste total de los recursos hardware ..... 815 €
Coste total de los recursos software ..... 2100 €
Coste total del material de oficina ..... 90 €

Coste total del material ..... 3005 €

El coste total de los materiales asciende a **tres mil cinco euros**.

VI.2. Coste de la mano de obra.

La realización de este proyecto ha sido llevada a cabo por las siguientes personas:

CONCEPTO	Coste hora	Total horas
1. Un Ingeniero de Telecomunicación redactor y director del proyecto	70 €	1500 horas
2. Un mecanógrafo encargado del mecanografiado del proyecto	12 €	100 horas

Coste total de la mano de obra ..... 106200 €

El coste total de la mano de obra asciende a **ciento seis mil doscientos euros**.

VI.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 ..... 3005 €
Coste total de la mano de obra ..... 106200 €

Coste total de ejecución de material ..... 109205 €

El presupuesto total de ejecución del material asciende a **ciento nueve mil doscientos cinco euros**.

VI.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 ..... 109205 €
22% gastos financieros, beneficios, etc... ..... 24025.1 €

Coste final de ejecución de material ..... 133230.1 €

El importe de ejecución por contrata asciende a **ciento treinta y tres mil doscientos treinta euros y diez centimos**.

VI.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 30.000 €	c = 1	7 %
Desde 30.000 €	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 133230.1 x 1 ..... 9326.11 €

Total derechos de visado ..... 9326.11 €

El importe total de los derechos de visado es de **nueve mil trescientos veintiseis euros y once centimos**.

VI.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	133230.1 €
Honorarios facultativos	9326.11 €
Presupuesto total	142556.21 €
16% de IVA	22809 €
Presupuesto final	**165365.21 €**

El importe total de este proyecto asciende a la cantidad de **ciento sesenta y cinco mil trescientos sesenta y cinco euros y veintiun centimos**.

En Alcalá de Henares, a 21 de Septiembre de 2004.

El Ingeniero de Telecomunicación

Fdo.: Eduardo González Maroto.

VII. Bibliografía

VII.1 Bibliografía básica

[Marrón - 00] *"Navegación Autónoma de una Silla de Ruedas en Interiores Parcialmente Estructurados".*
Marta Marrón Romera.
Escuela Politécnica UAH, Ingeniería en Electrónica.
Septiembre 2000.

[García - 01] *"Sistema de Posicionamiento y Autolocalización para Sillas de Ruedas Autónomas".*
Juan Carlos García García.
Escuela Politécnica UAH . Tesis Doctoral.
Diciembre 2001.

[López - 02] *"Automatización de un Edificio para la Navegación Autónoma de una Silla de Ruedas I"*
Pedro López Miguel.
Escuela Politécnica UAH . Ingeniería Técnica Industrial.
Junio 2002.

[Marrón - 02] *"Sistema de Posicionamiento Absoluto de un Robot Móvil. Fusión de Sensado Odométrico y Visión de Marcas Artificiales".*
Marta Marrón Romera.
Escuela Politécnica UAH, Trabajo de Investigación.
Junio 2002.

VII.2 Bibliografía de consulta

[Nuevo - 04] *"Robot telecomandado bajo entorno GNU/Linux".*
Jesús Nuevo Chiquero.
Escuela Politécnica UAH, Ingeniería de Telecomunicación.
Mayo 2004.

[Persson - 99] *"Communication Between PC and LON Using Enhanced Parallel Port".*
Johan Persson.
Diciembre 1999.

[Sebastián - 99] *"Guiado Semiautomático de una Silla de Ruedas".*
Eduardo Sebastián Martínez.
Escuela Politécnica UAH . Ingeniería en Electrónica.
Octubre 1999.

VII.3 Bibliografía de referencia

[Ceballos - 95] *"Curso de programación C / C++".*
Francisco Javier Ceballos.
Editorial Ra-Ma.
Año 1995.

[Sánchez - 02] *"Sistemas Operativos".*
Sebastián Sánchez Prieto.
Servicio de Publicaciones de la UAH.
2ª Edición, año 2002.

[Echelon - 97] *"LonWorks Technology Device Data".*
VV.AA.
Motorola.
Año 1997.

VII.4 Enlaces a la WWW

- Video 4 linux 2

  http://linux.bytesex.org/v4l2/

- Documentación sobre POSIX y Tiempo Real:

  http://atc1.aut.uah.es/~arqui/

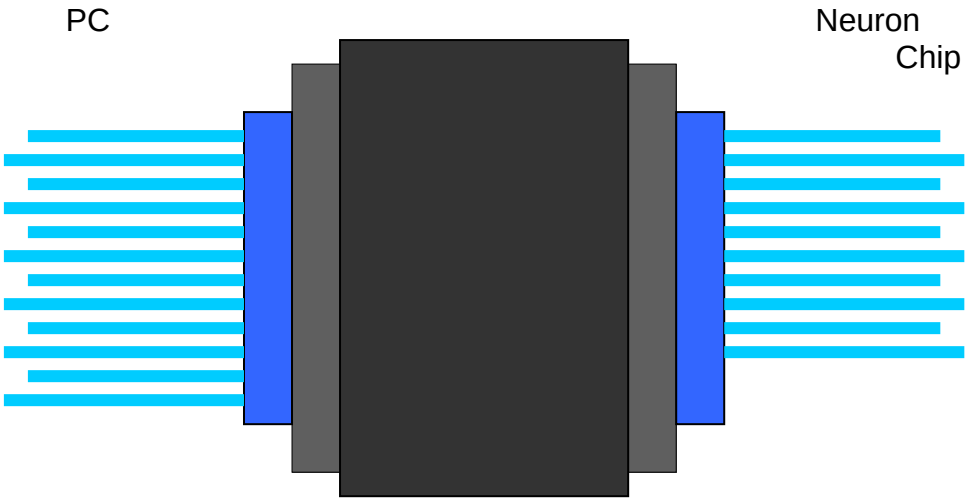
- Depurador de gestion de memoria Valgrind

  http://valgrind.kde.org/

- Proyecto de documentación de Linux (TLDP)

http://www.tldp.org

Correspondencia de líneas entre el puerto paralelo y la conexión al Neuron Chip.



Lado PC	Lado Neuron
2	3
3	14
4	4
5	15
6	5
7	16
8	6
9	17
14	7
15	8
17	18
23	23

La numeración corresponde a la de las líneas vistas desde los terminales externos del conector diseñado, y la tabla expresa el conexionado realizado dentro de la caja.