Manual del robot EyeBot

Jose M. Cañas, Esther García, Vicente Matellán
Universidad Rey Juan Carlos
jmplaza@gsyc.escet.urjc.es

TR-GSYC-2002-1, versión 2.5

Resumen

Eyebot es un robot móvil en el que vamos a desarrollar nuestras investigaciones orientadas principalmente a la creación de un equipo de fútbol para participar en la Robocup. El robot dispone de una serie de sensores y actuadores para que pueda desevolverse en el entorno.

Dispone de tres sensores de infrarrojos, una cámara digital y dos encoders, uno en cada motor. Además dipone de dos motores que permiten el movimiento del robot y dos servos, uno para mover la cámara y el otro para el pateador.

Eyebot tiene un microprocesador Motorola 68332, con 512 KB de Flash-ROM y una memoria RAM de 1 MB, entre otra cosas. Además cuenta con un módulo de radio para comunicarse entre distintos robot y para comunicarse con un PC.

Este manual es una guía para el programador de aplicaciones en el robot. Describe brevemente los sensores y los actuadores del robot, el software de acceso a los distintos recursos hardware, distintas librerías de apoyo.

Índice

Índice de cuadros

Agradecimientos

Este manual ha sido realizado por los alumnos del grupo de robótica de la URJC. Contribuciones importantes se deben a ellos: Esther García Morata, Félix San Martín de la Fuente, Carlos Agüero, Victor Gómez, Pedro José Macedo, Manuel Peño, Ana Martínez, María Angeles Crespo.

1. Introducción

El Eyebot es un robot móvil sobre el que estamos llevando a cabo una serie de investigaciones, centradas sobre todo en la Robocup (competición de fútbol con robots). Este

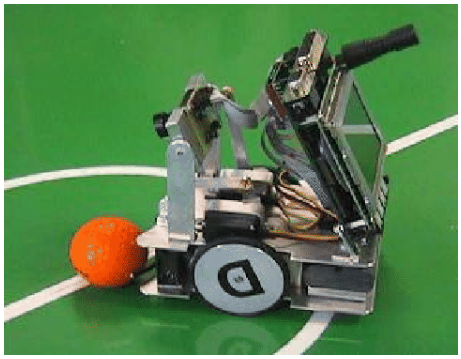

Figura 1: EyeBot

robot dipone de ciertos sensores que le permiten moverse por el espacio de forma autónoma, además dispone de una cámara que le facilita la tarea del posicionamiento y un 'kicker' para dar al balón.

Este manual nace con la idea de servir de ayuda al programador de aplicaciones para el robot Eyebot, describiendo el entorno de programación que tiene a su disposición el investigador para materializar en el robot sus propuestas, además se proporciona un mapa del mismo para saber qué sensores tiene, sus caraterísticas, y qué puede realizar con ellos. Una parte importante de este entorno la constituyen las librerías hechas por el fabricante para acceder a distintos sensores y actuadores. Otra parte significativa la forma el conjunto de programas ya desarrollados dentro del grupo y los servicios que ofrecen.

La documentación de referencia sobre el robot y su entorno la ofrece directamente el fabricante en la página oficial del Eyebot[1].

Para generar los programas a descargar en el robot usamos un compilador cruzado gcc68 sobre una plataforma Linux. El lenguaje usado para programar al robot es C, pudiendo ser también utilizado el lenguaje ensamblador del M68000.

El sistema operativo sobre el cual corren todas las aplicaciones es linux, un sistema similar a Unix, de libre distribución, multitarea y multiusuario, de 32 bits muy robusto y ágil. C es el lenguaje elegido por el fabricante para codificar sus librerías y la mayoría del software dentro del grupo ha sido programado también en lenguaje C. La distribución de Linux utilizada incluye gratuitamente *gcc*, un compilador C/C++ para ese sistema operativo.

En su contra tiene que no es un sistema de tiempo real, pues no permite acotar plazos. Sin embargo ha resultado suficiente para los requisitos de rapidez necesarios en nuestras aplicaciones. Linux ofrece las librerías necesarias para desarrollar interfaces gráficas en el sistema más extendido en el mundo Unix, X-Window.

[1] `http://www.ee.uwa.edu.au/~braunl/eyebot/`

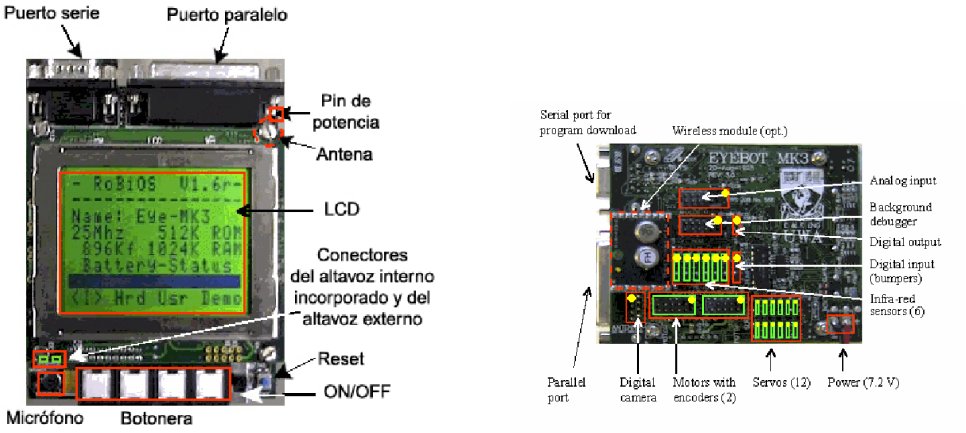

Figura 2: Vistas frontal y trasera del EyeBot

2. El robot eyebot

2.1. Hardware

El robot está constituido por un microprocesador que es el encargado de ejecutar los distintos programas, distintos sensores como son los dos encoders, uno para cada motor, que se encargarán de detectar el movimiento del robot y tres infrarrojos (en adelante PSD, Position Sensitive Detector), que detectarán la presencia de un obstáculo cercano. Una cámara digital situada en la parte frontal del mismo capturará imágenes del entorno en el que se desenvuelva el robot.

Los actuadores conectados al Eyebot son dos motores, que se encargarán de mover las ruedas y dos servos (uno para la cámara y otro para el pateador).

A través de un puerto serie se pueden descargar los distintos programas al Eyebot, este puerto también puede ser utilizado para comunicar el robot con un ordenador. Esta comunicación también es posible utilizando el módulo de radio, el cual también permite la comunicación entre distintos robots. Un puerto paralelo es utilizado para el depurador externo.

Gracias a una pantalla (en adelante LCD, Liquid Cristal Display) es posible visualizar lo que está ocurriendo en el robot. En la parte inferior hay 4 botones que permiten la interacción del usuario con el programa. Adicionalmente lleva incluido un micrófono que permite capturar sonidos con la capacidad de poderlos reproducir gracias a un altavoz interno.

En las siguientes imágenes se ilustra gráficamente la integración física de cada componente en la placa principal del Eyebot.

Todos los recursos hardware del robot se pueden agrupar en varias categorías que se

analizarán a continuación.

2.1.1. Procesamiento

- Microprocesador
  Se trata de un microprocesador Motorola 68332 a 35 MHz con una memoria Flash-ROM de 512 KB que son para el SO y los programas de usuario, tiene una memoria RAM de 1 MB que permite ejecutar los programas almacenados en la ROM. Dispone de 2 puertos serie, 1 paralelo, además tiene 8 entradas digitales, 8 salidas digitales y 8 entradas analógicas. Puede controlar 2 motores con sus correspondientes encoders, 12 servos, una cámara y 6 sensores de infrarrojos(PSD).

2.1.2. Sensores

Los elementos encargados de recoger información del entorno para su posterior tratamiento son los infrarrojos, los encoders y la cámara.

- Infrarrojos
  Los tres infrarrojos (GP2D02 de Sharp) miden la distancia a un obstáculo cercano y están situados uno a cada lado del robot y otro en el frontal, con la peculiaridad de que el situado en el lado izquierdo está en posición inversa a los otros dos. Su principio de funcionamiento se basa en emitir luz infrarroja y medir la cantidad de energía que rebota. Si reciben mucha es que el obstáculo esta muy cerca. Según sus hojas de características su rango va de los 10 a los 80 cm. Se han determinado empíricamente sus errores:

  Error angular: 5-7^{o} aprox.

  Error radial eyebot:

  Error Frontal: 19 %.

  Error Izquierdo: En posición inversa al derecho, da un error superior al 50 %.

  Error Derecho: 42 %.

- Encoders
  Dos encoders (uno para cada motor) devuelven un número de pulsos, los cuales son indicativos del desplazamiento que cada rueda ha realizado. Tienen una resolución de un cuarto de vuelta. El número de pulsos que devuelve cada encoder tras un desplazamiento de un metro está definido en la HDT.

- Cámara
  La cámara incluida trabaja con 24 bits en color o en escala de grises, proporcionando una resolución de 80x60 pixels. Esta resolución es suficiente para la mayoría de las tareas que realiza el robot y permite un procesamiento rápido de la imagen.

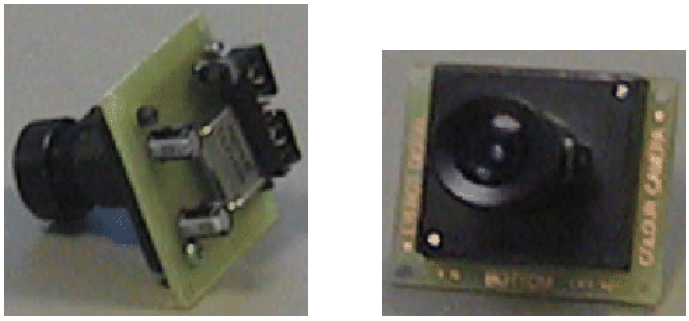

Figura 3: Vistas posterior y frontal de la cámara

Esta cámara puede ser usada sobre el eyebot directamente o mediante un adaptador conectarla al PC, utilizando el software adecuado (Improv).

Lleva un servo asociado que le permite un giro horizontal o vertical, dependiendo del modelo, con el fin de facilitar la visión de los objetos deseados.

2.1.3. Actuadores

- Motores de continua

  Dos motores de continua posibilitan la movilidad del robot. La velocidad de los motores se fija en porcentajes, con un rango de valores desde -100 a 100, negativo hacia atrás y positivo hacia delante. 0 detiene el motor.

  El patillaje destinado al motor permite que éste pueda ser de las marcas Faulhaber, MiniMotor y MicroMo.

- Servos
  El Eyebot dispone de dos servos uno para mover la cámara y otro para el pateador. Un servo es un motor controlado por pulsos PWM. A diferencia de los motores de continua estos pueden ser posicionados y enclavados en un ángulo determinado. Los valores máximos para el ángulo y velocidad de giro pueden ser activados por software. Para el modelo HiTech HS81 utilizado el ángulo máximo y mínimo y el máximo ángulo están determinados por los pulsos PWM de 0.74 ms y 2.14 ms.Estos valores están representados en el HDT dentro de la descripción de los servos.

2.1.4. Comunicaciones

- Puerto Serie
  A través de un conector RS-232 situado en la parte frontal del robot es posible conec-

tar el Eyebot a un PC, a un Mac (con adaptador especial) o a una workstation. De este modo se posibilita la descarga de programas o el envío de comandos, datos y medidas entre distintas plataformas. Estas transmisiones se pueden realizar a diferentes velocidades, 9600, 19200, 38400, 57600 y 115200 baudios.

- Puerto Paralelo
  El puerto paralelo se encuentra al lado del puerto serie. Se utiliza para conectar el Eyebot con el ordenador y poder realizar una depuración del programa con las herramientas que suministra Motorola para ello.

- Módulo de radio
  Este módulo permite la comunicación entre robots sin la necesidad de un host (ordenador central),también permite la comunicación de un robot con un PC, así éste puede hacer un control remoto de los robots o monitorizarlos, todo esto vía radio. El módulo de radio se conecta a uno de los puertos serie del robot. Esta red trabaja con token ring virtual (paso de testigo). Características técnicas: Frecuencia de trabajo a 433MHz, velocidad máxima de transmisión de 9600 baudios, se incluye un protocolo de tolerancia a fallos, transmisión de 8 bits.

2.1.5. Interacción

Dentro de esta categoría se incluyen aquellos elementos que posibilitan la interacción del usuario con el sistema.

- LCD
  La LCD consta de un grid de 128 x 64 pixels, desde el que se pueden visualizar hasta 17 caracteres ASCII por línea, con un total de 8 líneas en total (la línea inferior está reservada para las etiquetas de menú.

- Botonera
  En la parte inmediatamente inferior a la LCD hay una hilera de 4 botones cuya funcionalidad dependerá del programa o actividad que se esté ejecutando en cada momento.

- Altavoces
  En la parte frontal del Eyebot hay dos conectores para distintos altavoces. Uno de ellos viene integrado con el Eyebot, se trata de un altavoz del tipo piezo-eléctrico. En el otro conector es posible el uso de un altavoz externo estándar de 8 Ohm. El volumen puede ser ajustado con un potenciómetro situado en la parte inmediatamente inferior a los conectores.

- Micrófono
  Un micrófono en miniatura viene integrado en la parte frontal del Eyebot. A través de él es posible recoger sonidos externos e integrarlos en los programas.

Figura 4: Display y Botonera

2.2. Sistema operativo: RoBIOS

Cuando se enciende un Eyebot, sea del tipo que sea, en él se está ejecutando un sistema operativo propio. Este sistema operativo se llama RoBIOS (Robot Basic I/O System) y consta de tres elementos: la **consola** que interactúa con el usuario a través de la pantalla y los botones; la **tabla HDT** con los dispositivos hardware conectados; y la **interfaz de programación** que ofrece a los programadores.

El sistema operativo gestiona los diferentes recursos del sistema y muestra al usuario humano una consola a través de la que se puede interactuar con el robot. Desde ella se puede cargar y ejecutar programas, se pueden hacer comprobaciones de funcionamiento de los sensores y motores del robot, se pueden ejectuar demostraciones, etc.

Dentro del sistema operativo hay una tabla, llamada HDT (Hardware Description Table), en la que se definen los diferentes dispositivos hardware realmente conectados al robot concreto. Todos los Eyebot disponen del mismo RoBIOS pero diferente HDT dependiendo ésta de los dispositivos conectados.

Para facilitar el acceso a los distintos recursos desde los programas del usuario RoBIOS ofrece una interfaz de programación. Esta interfaz es un conjunto de funciones para acceder y manipular los distintos sensores, actuadores y resto de dispositivos del robot. Estas funciones están escritas en C y se enlazan con el código del programa usuario, de manera que resulta sencillo acceder a los distintos elementos. Sobre las funciones del sistema operativo el fabricante proporciona un conjunto de librerías útiles que enriquecen la interfaz para el usuario programador del eyebot.

En esta sección se describe el uso de la consola para realizar test al Eyebot, preparar

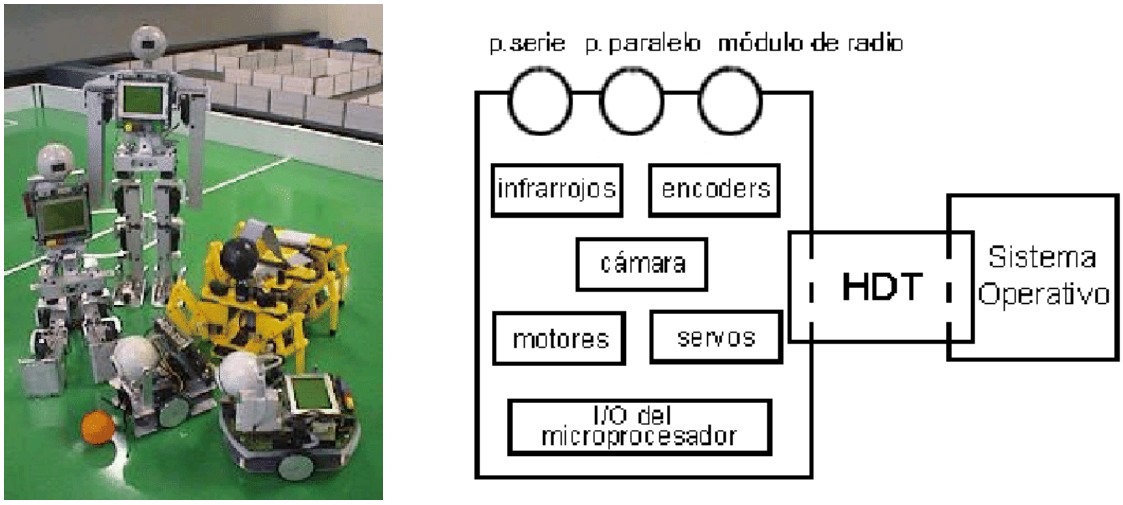

Figura 5: Ejemplos de robots con la misma RoBIOS y distinta HDT (izda), HDT (derecha)

la descarga de programas, así como ejecutarlos. También se explica la tabla de descripción del hardware, con un ejemplo de cómo añadir un nuevo componente al sistema. La interfaz de programación será tratada en un apartado propio, dado su tamaño y su importancia.

2.3. Consola

A través de una consola el sistema operativo ofrece la posibilidad de realizar test internos a los dispositivos hardware del Eyebot, de cargar los programas del usuario, y de ejecutarlos. También es posible ejecutar distintos programas de demostración del funcionamiento del sistema.

El funcionamiento de la consola se realiza mediante menús en los que el usuario va eligiendo opciones en pantallas pulsando botones.

Test interno. Para realizar un test de los dispositivos hardware hay que seguir los pasos siguientes:

- Seleccionar Hrd/HDT desde el menú principal del Eyebot.

Se llega a una pantalla en la que aparecen todos los dispositivos hardware que están introducidos en la HDT. Dentro de esta pantalla, pulsando la tecla “Tst” se pasa a diferentes pantallas donde se realizan los test de comprobación. Por esta pantalla es posible cambiar de tipo de dispositivo con la tecla “Nxt”. Con la tecla “+” se puede seleccionar entre los distintos dispositivos de un mismo tipo, como por ejemplo entre los 3 infrarrojos, o entre

Figura 6: Pasos para llegar al test de los dispositivos

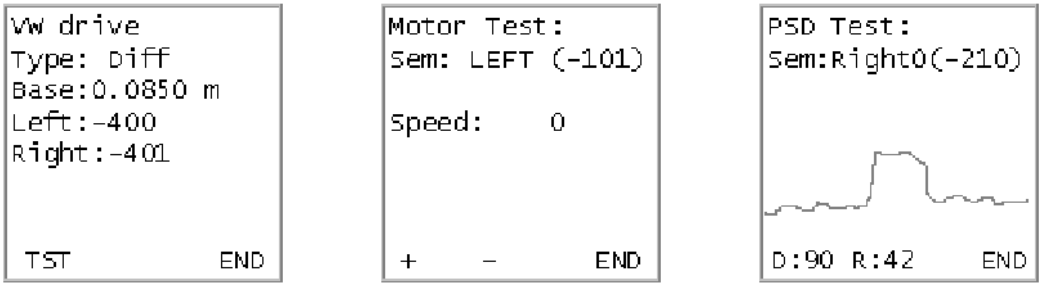

Figura 7: Test del control VW (izda), del motor izquierdo (centro) y del encoder (derecha)

los 2 servos, etc. El número de dispositivos de un mismo tipo está indicado por el número al principio de cada tipo.

- Una vez que se ha seleccionado el dispositivo deseado se pulsa la tecla “Tst”. Para cada tipo de dispositivo la pantalla de test que aparece es diferente.

- Test del control VW.
  El test del control VW que se puede realizar consiste en el desplazamiento del robot una distancia total de 0.085 m (el parámetro Base de la pantalla). Este desplazamiento estará dividido en 4 pasos. Un avance de la mitad de la distancia, un giro de 180 grados, un avance de la otra mitad de distancia y otro giro de 180 grados que coloca al robot en el punto de partida. Para realizar el test en esta pantalla sólo hay que pulsar el botón “TST”. El test finaliza pulsando el botón “END”.

- Test de los motores.
  Este test se realiza para cada motor de manara independiente. El motor seleccionado (LEFT o RIGHT) se indica en la pantalla, junto con el valor de la velocidad (Speed). Este valor es inicialmente de 0 y puede ser variado en incrementos de 10 en 10 con las teclas "+" y "-". El motor seleccionado comenzará a girar a la velocidad indicada (velocidad en porcentaje). Los valores máximo y mínimo para la velocidad son de 100 y -100 respectivamente, siendo los positivos movimientos hacia delante y los negativos hacia atrás. El test acaba pulsando la tecla "END".

- Test para los encoders.
  Cada encoder va asociado a su correspondiente motor, lo cual se indica en la pantalla, donde aparecerá en encoder LEFT o RIGHT junto con el motor correspondiente.

  Para realizar el test hay que aumentar o disminuir la velocidad (inicialmente 0) del motor al que está asociado el encoder con las teclas "+" y "-". Cuando el robot se desplace en el valor Ticks/s de la pantalla aparecerá el desplazamiento en ticks del motor correspondiente desde la medida anterior, (el incremento). Por tanto si el motor no se mueve, el valor que aparece aquí es 0. En el valor Counter aparece el desplazamiento total del motor desde que se comenzó el test. Este valor es posible resetearlo con el botón "RST". Para finalizar el test hay que pulsar el botón "END".

- Test de los servos.
  Por defecto los servos que vienen con el Eyebot son el de la cámara, identificado en la HDT como "Cam 11" y el del pateador, identificado como "Kick12".

  El test de estos servos parte de una posición inicial de 128. A partir de aquí con los botones "+" y "-" se puede cambiar esta posición en incrementos de 8 unidades. El rango posible para los valores de la posición es de 0 a 255, pudiendo alcanzarse el valor máximo pulsando el botón "max". Para finalizar el test se deberá pulsar la tecla "END".

- Test para los infrarrojos.
  Los infrarrojos que vienen de serie con el Eyebot están identificados semánticamente en la HDT como "Right0", "Front1" y "Left 2". Para cada uno de ellos es posible realizar este test. En él aparece el nombre del infrarrojo seleccionado, y en la parte de abajo dos valores identificados como "D" y "R". El primero indica la distancia en milímetros para la cual el sensor en cuestión está detectando un obstáculo. Este valor no es la salida del sensor, sino este valor modificado por una tabla de calibración incluida en la HDT. El valor de salida del sensor aparece bajo el identificador "R" (Raw).

  En la pantalla aparece un trazo continuo que se modifica en función de las medidas del sensor, estableciendo una referencia de la distancia que está midiendo. La finalización de este test se produce al pulsar la tecla "END".

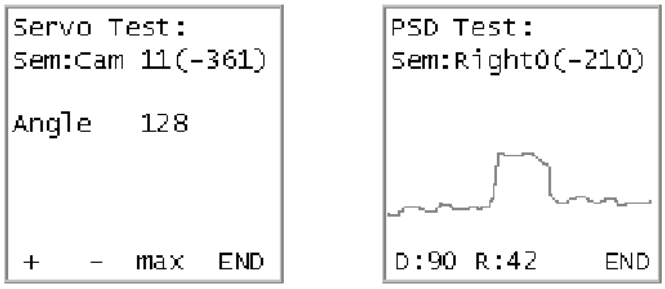

Figura 8: Test del servo asociado a la cámara (izda) y del infrarrojo derecho (derecha)

Figura 9: Pasos para llegar al menú de configuración del puerto serie

Carga y ejecución de programas. Además de un testeo de los elementos hardware conectados al robot también es posible hacer configuraciones sobre los distintos puertos del sistema en cuanto a la velocidad y el modo de transmisión, de manera que se puedan realizar descargas de los programas del usuario desde el PC al robot. Cuando un programa ha sido cargado se encuentra en la RAM del sistema operativo. Si el robot se apaga el programa se pierde, para solucionar este programa es posible almacenarlo en la ROM del sistema, de manera que puede ser ejecutado cuando el usuario desee.

Los pasos a seguir para efectuar la descarga de un programa al robot son:

- Seleccionar Hrd/Set/Ser desde el menú principal del Eyebot. En la pantalla que aparece es posible configurar el interfaz de descarga (en este caso será el SERIAL1), la velocidad de transmisión (115200 Baudios) y, si se desea, la visualización de los bytes transmitidos.
- Una vez configurados los parámetros deseados se pulsa “End” hasta alcanzar el

Figura 10: Descarga de un programa

menú principal, desde donde se pulsa Usr/Ld para poner el Eyebot en modo recepción de programas.

- Enviar el programa desde el ordenador utilizando el script dl (dl demo.hex). En este script se indica el puerto donde estará conectado el cable serie que lo une con el Eyebot. También se indica la velocidad a la que se ha de enviar que ha de coincidir con la seleccionada en el Eyebot en el punto anterior. En la pantalla aparecerá el nombre del programa que se está descargando, y si se ha activado la posibilidad también aparecerá el número de bytes que se han transmitido.
- El Eyebot indica cuando se ha completado la transmisión, tras lo cual el programa puede ser ejecutado pulsando "Run". En caso de producirse algún tipo de error durante la recepción del programa un mensaje por pantalla avisa de esta situación.
- El botón "Rom" permite almacenar en la ROM los programas descargados (hasta un máximo de tres programas), para ello hay que situar el indicador (*) sobre la posición deseada y pulsar "Sav". Una vez almacenados será posible cargarlos en la RAM para su ejecución en cualquier momento (entrando en este menú, seleccionando el programa de la ROM y pulsando "Ld").

Ejecución de demos. Desde la pantalla principal se puede pulsar la tecla "Demos" de manera que se entra en un programa de demostración del funcionamiento de los distintos elementos. El fabricante proporciona tanto el ejecutable como el código fuente de este tipo de programas. Por defecto un programa de demostración viene cargado cuando se compra un Eyebot y puede ser diferente en función de la versión de RoBIOS.

Figura 11: Manejo de los programas almacenados en la ROM

2.4. HDT

A un mismo microprocesador se le pueden conectar diferentes elementos físicos, ya sean sensores, actuadores, etc. Por tanto se puede decir que el robot es un sistema abierto, ya que permite una total configuración de la arquitectura del mismo. La parte del sistema operativo donde se configura el soporte para el distinto hardware conectado se llama HDT. Desde esta tabla se da nombre a todos los recursos del sistema. Estos nombres serán referenciados desde la interfaz de programación para acceder a un determinado recurso.

El HDT (Hardware Description Table) es un concepto que permite a los controladores hardware detectar y usar de una manera relativamente sencilla el hardware conectado al Eyebot. Principalmente consta de dos partes: Procedimientos de acceso y estructuras de datos.

- Procedimientos de acceso.
  Se trata de rutinas que forman parte del RoBIOS y se utilizan para comprobar si un componente incluido en el fichero de estructuras de datos, está físicamente integrado en el sistema. Estas rutinas son internas y no pueden ser utilizadas por los programadores.

- Estructuras de datos.
  Distintos Eyebot o distintas configuraciones hardware en un mismo Eyebot necesitan su propio fichero HDT, el cual contiene toda la información sobre el hardware del que dispone el Eyebot y de cómo acceder al mismo. Esta información está recogida en las estructuras de datos.

Cuando se desee añadir un nuevo hardware se ha de modificar la HDT para indicar al sistema operativo este cambio.

Cada componente que se añada tiene una estructura definida en el fichero de cabecera <hdt.h>. Esta estructura habrá que tenerla en cuenta a la hora de modificar la HDT. Para

realizar los cambios oportunos se deberá editar el fichero que se esté utilizando como HDT hasta el momento. En él se ha de incluir el nuevo componente, respetando la estructura definida en el fichero de cabecera, (ver ejemplo).

Una vez incluida la estructura del componente se ha de añadir a la lista de componentes activos, que no es más que una matriz en donde están todos los componentes del sistema.

Por ejemplo, supongamos que se desea añadir otro servo. Para ello en el fichero <hdt.h> vemos que la estructura de un servo es la siguiente:

```
typedef struct
{
  int     driver_version;
  int     tpu_channel;
  int     tpu_timer;
  int     pwm_period;
  int     pwm_start;
  int     pwm_stop;
}servo_type;
```

En esta estructura driver_version es la versión del driver para el cual el nuevo servo es compatible. Para saber el número de versión, se deberá consultar la documentación referente al RoBIOS, ya que este valor puede verse modificado de una versión a otra del sistema operativo. Si no se hace referencia, se utilizará la versión que estén utilizando los otros servos conectados.

El valor de tpu_channel dependerá del conector físico en la placa base al que se desee conectar el nuevo servo. Los valores posibles son 0 ... 15.

Para determinar el valor de tpu_timer hay que tener en cuenta que un servo necesita recibir una señal PWM para poder anclarlo en diferentes posiciones. El temporizador interno del microprocesador es capaz de generar esta señal y enviarla al canal indicado (tpu_channel). Los valores posibles para indicar aquí son TIMER1 y TIMER2. La elección de uno u otro dependerá del periodo que se desee para la señal PWM. La velocidad del TIMER1 para una velocidad de CPU de 33MHz es de 4.25 MHZ, y la del TIMER2 es de 0.512 MHz. Para otras velocidades de CPU puede ser calculada utilizando las fórmulas:

TIMER1[MHz] = 4MHz * (16MHz + (CPUclock[MHz] % 16))/ 16

TIMER2[MHz] = 512KHz * (16MHz + (CPUclock[MHz] % 16)) / 16

El valor del pwm_period está determinado por el periodo de la señal PWM que necesita el servo en microsegundos. Un servo normal necesita un periodo de 20000us, por ello es preferible utilizar el temporizador TIMER2 como valor del tpu_timer, ya que con un mayor período del temporizador se obtienen los suficientes intervalos discretos para posicionar el servo en la posición exacta.

Pwm_start es el tiempo mínimo a la alta del periodo de la señal PWM. Los valores posibles van desde 0 hasta el valor pwm_period. Normalmente el servo necesita un valor de pwm_start de 0.7ms (700us).

Pwm_stop es el tiempo máximo a la alta del período de la señal PWM. Los valores posibles también van desde 0 hasta pwm_period. El valor normal es de 1.7ms (1700us).

Estos dos valores son utilizados para determinar los ángulos máximo y mínimo a los que se puede anclar el servo, si es que se desea que no tenga todo el recorrido. El valor pwm_stop puede ser mayor que el de pwm_start y viceversa, dependiendo del sentido de la rotación que se desee.

Cuando se han determinado los valores con los que se desea rellenar la estructura del nuevo servo se modifica el fichero htd.c que se esté utilizando actualmente. Junto con los otros servos se ha de incluir la línea:

servo_type servoN={driver_version, tpu_channel, tpu_timer, pwm_period, pwm_start, pwm_stop};

Donde N es el identificador elegido para el servo conectado.

En el mismo fichero, hay que buscar el array HDT_entry_type HDT[] y añadir el nuevo hardware a la lista.

```
HDT_entry_type HDT =
 {
    ...
    ...
    { SERVO,SERVON,"Nuevo",(void *)&servoN},
    ...
    ...
 };
```

Donde N es el identificador elegido para el servo conectado.

Para cualquier hardware que se desee añadir hay que rellenar con cuidado los valores de su estructura, ya que en función de estos valores el sistema se comportará correctamente o no. Por ejemplo, para incluir un encoder, uno de los valores que hay que rellenar es el número de pulsos que éste devuelve por cada metro recorrido por el robot. Este valor es muy importante, ya que la función de los encoders es determinar los cambios de posición del robot, y si el parámetro está mal ajustado las medidas no serán correctas.

En la HDT también es posible incluir tablas de calibración para aquellos sensores que lo necesiten. Un ejemplo de estos sensores son los infrarrojos (PSD). Estos sensores presentan variaciones entre la distancia a la que están midiendo un obstáculo y la distancia real a la que éste se encuentra. Para modificar estas tablas hay que realizar medidas empíricas que ayuden a determinar los parámetros correctos.

Una vez que se ha modificado el fichero fuente hay que compilarlo. Para ello se utiliza el script *gcchdt* suministrado por el fabricante (Uso: *gcchdt hdtfile.c*). El resultado es un fichero con extensión ".hex". Este fichero deberá ser descargado al Eyebot a través del puerto serie. Automáticamente el Eyebot reconocerá que se trata de una nueva configuración hardware, por lo que sustituye la anterior con ésta. Tras la sustitución será necesario resetear el Eyebot.

3. Interfaz de programación

3.1. Acceso a los sensores

El RoBIOS ofrece distintas posibilidades a la hora de acceder a los infrarrojos, los encoders y la cámara en función del uso que se vaya a hacer de estos recursos.

3.1.1. Infrarrojos PSD

Acceder a los sensores de infrarrojos PSD (Position Sensitive Detector) únicamente requiere la inicialización previa de los mismos con la función PSDInit(nombre_psd_en_HDT). Una vez hecho esto es posible obtener los valores medidos por cada uno de ellos a través de la función PSDGet(sensor).Es importante tener en cuenta que el valor que devuelve esta función es distinto al valor real que está midiendo el infrarrojo (se puede obtener con PSDGetRaw(sensor)). Esta diferencia es debida a que se está trabajando con sensores cuya respuesta está muy condicionada al entorno en el que se hayan, y por ello es necesario la utilización de una tabla de calibración para mitigar los errores. Esta tabla se encuentra en la HDT y para modificarla hay que modificar la misma y volver a cargarla al Eyebot. Es conveniente la liberación del recurso al finalizar el programa, PSDRelease() libera todos los infrarrojos que se estén utilizando.

3.1.2. Acceso a los encoders

Para acceder a los encoders es necesaria su inicialización previa con la función QuadInit (nombre_encoder_en_HDT), tras lo cual será posible resetearlos (QuadReset (encoder)) y realizar lecturas de los mismos con la función QuadRead (encoder).

Pero hay que tener en cuenta que si se está utilizando el control VW esta manera de acceso no es correcta, ya que este tipo de control inicializa los encoders internamente, por lo que si los encoders ya estaban inicializados el control VW no funcionará , y si no lo estaban y tras inicializar el control VW tratamos de inicializarlos, no se devolverán los manejadores correctos, y por tanto las posteriores lecturas que necesiten estos manejadores no funcionarán. La manera de acceder a los mismos será utilizando los manejadores que devuelve la inicialización interna. Estos valores no son arbitrarios y están definidos por software, siendo para el encoder izquierdo 0x0c02 y para el derecho 0x0304. Por ejemplo, la lectura del valor del encoder derecho se realizaría así: `QuadRead(0x0304)`.

Otro aspecto a tener en cuenta es el valor que devuelven dichos encoders, que son pulsos. En la HDT está definido el número de pulsos que devuelve cada encoder por metro. Este valor también es modificable por el usuario tras las calibraciones necesarias. Es conveniente liberar el recurso en la finalización del programa a través de la función `QuadRelease (encoder)`.

Formato Función	*P.Entrada*	*P.Salida*	*Descripción*
PSDHandle PSDInit(DeviceSemantics semantics)	(semantics) Nombre del PSD deseado (ver hdt.h)	Manejador del PSD para las demás operaciones	Inicializa el PSD deseado. Pueden ser inicializados hasta 8 PSDs
int PSDRelease(void)	Ninguno	Ninguno	Detiene el funcionamiento de todos los PSDs inicializados
int PSDStart (PSDHandle bitmask, BOOL cycle)	(bitmask) Lista de los manejadores de los PSDs a los que se le aplicará la función. (cycle) TRUE = medida continua. FALSE = medida simple.	-1 = error manejador incorrecto 0 = Ok 1 = ocupado (ya esta siendo utilizado)	Comienza la medida en forma continua o simple de los PSDs especificados en la entrada. En forma continua los sensores toman una nueva medida cada 60ms.
int PSDStop(void)	Ninguno	Ninguno	Para la medida de los PSDs después de completarse ésta
BOOL PSDCheck(void)	Ninguno	TRUE = si un resultado válido está disponible	Testea si el resultado de una medida en un PSD es válido.
int PSDGet (PSDHandle handle)	(handle) Manejador del PSD deseado. 0 para ver el tiempo actual del ciclo de medida del sensor.	Distancia real en mm (conversión interna a través de una tabla)	Devuelve el valor del tiempo del ciclo de medida o la distancia medida por el PSD seleccionado. Si la lectura del lector está fuera de rango la función retorna PSD_OUT_OF_RANGE (=9999)
int PSDGetRaw (PSDHandle handle)	(handle) Manejador del PSD deseado. 0 para ver el tiempo actual del ciclo de medida del sensor	Valor en crudo del sensor (sin conversión)	Devuelve el valor del tiempo del ciclo de medida o el valor en crudo del PSD seleccionado.

Cuadro 1: Funciones para los PSD

Formato Función	*P.Entrada*	*P.Salida*	*Descripción*
QuadHandle QuadInit (DeviceSemantics semantics)	(semantics) Nombre del encoder deseado.	Manejador del encoder ó 0 si hay algún error.	Inicializa el encoder especificado. (Pueden ser inicializados hasta 8 encoders).
int QuadRelease (QuadHandle handle)	(handle) Lista de los manejadores de los encoders a los que se les aplicará la función.	0= ok -1 = manejador incorrecto	Detienen el funcionamiento de los encoders especificados.
int QuadReset (QuadHandle handle)	(handle) Lista de los manejadores de los encoders a los que se les aplicará la función.	0= ok -1 = manejador incorrecto	Reinicia uno o mas encoders.
int QuadRead (QuadHandle handle)	(handle) Manejador del encoder deseado	Valor del contador (32bits 0 a 2$\hat{}$32-1) Si el manejador es incorrecto el resultado del contador será 0	Lee el valor real del contador del encoder.
DeviceSemantics QUADGetMotor (DeviceSemantics semantics)	(handle) Manejador del encoder deseado	Semántica del motor correspondiente. 0 = manejador incorrecto	Devuelve la semántica del motor correspondiente
float QUADODORead (QuadHandle handle)	(handle) Manejador del encoder deseado	Metros desde la última reinicialización del encoder.	Devuelve la distancia desde el último punto de reinicialización. No es el número de metros desde la última reinicialización, es el número de metros recorridos desde el punto de salida (fijado por la siguiente función).
int QUADODOReset (QuadHandle handle)	(handle) Lista de los manejadores de los encoders a los que se les aplicará la función.	0= ok -1 = manejador incorrecto	Reinicializa los encoders definiendo el punto de salida.

Cuadro 2: Funciones para los encoders

3.1.3. Imágenes de la cámara

El procedimiento de acceso a la cámara es sencillo, sólo hay que inicializarla a través de la función `CAMInit(zoom)` indicándole el factor de zoom que se desea utilizar (WIDE,NORMAL o TELE). Tras la inicialización es posible la captura de imágenes.

Se pueden obtener imágenes en escala de grises utilizando la función `CAMGetFrame(imagen)` o imágenes en color con la función `CAMGetColFrame (imagen,color)`. La variable color es un entero: 0 -¿captura la imagen en color (24 bits). 1 -¿la convierte automaticamente en escala de grises (4 bits). Divide la imagen en tres más pequeñas, correspondientes a lo s tonos rojo, verde y azul.

También se puede hacer esta conversión posteriormente con la función: IPColor2Grey(..).

Hay que tener en cuenta que si lleva un tiempo sin utilizarse las primeras imágenes que se obtendrán serán de poca calidad, muy claras y sin apenas contraste. Esto es debido al transitorio que aparece en la inicialización. La máxima velocidad de captura es de 3.78 imágenes por segundo.

Una vez que está inicializada es posible acceder para consultar y/o modificar los va-

lores de configuración, como el brillo, el contraste, el offset, etc.., gracias a la función `CAMSet(brillo,offset,contraste)`.

Es posible también seleccionar si se desea ajustar la luminosidad de modo automático a través de `CAMMode (AUTOBRIGHTNESS)` o `CAMMode (NOAUTOBRIGHTNESS)`.

Es conveniente que en la finalización del programa se libere el recurso a través de la función `CAMRelease()`.

Ejemplos:

```
/* Programa para el manejo del LCD y la cámara del Eyebot */
/* Captura la imagen en blanco y negro y la saca por LCD*/

#include <eyebot.h>

image imagen;

int main(void)
{
     CAMInit(NORMAL); /*Inicializa la cámara con zoom normal */
     LCDClear();    /*Borra pantalla*/
     LCDMenuI(4,"FIN"); /*Muestra botón de fin*/
     do
       {
CAMGetFrame((image*)&imagen); /*Captura la imagen*/
LCDPutGraphic((image*)&imagen);/*Imprime imagen por el LCD */
     }while (KEYRead() != KEY4);
     CAMRelease(); /*Desactiva la cámara*/
   return 0;
}
```

3.2. Acceso a los actuadores

3.2.1. Control en lazo abierto de los motores

A través de las funciones suministradas por el fabricante es posible acceder a los motores y controlarlos de dos maneras distintas: el acceso directo a los mismos y el acceso a través del control VW.

Para acceder directamente es necesaria una inicialización previa utilizando la función MOTORInit(nombre_motor_en_HDT). Tras esto se puede indicar al motor deseado que se desplace con la función MOTORDrive(motor,velocidad). La velocidad será positiva si se desea un desplazamiento hacia delante y negativa si se desea hacia atrás.

Formato Función	*P.Entrada*	*P.Salida*	*Descripción*
CAMInit (zoom)	Factor de zoom (WIDE,NORMAL,TELE).	Versión de la cámara o código de error: 255= cámara no conectada. 254= error al inicializar. 0-15= cámara en blanco y negro. 16-31= cámara en color.	Resetea e inicializa la cámara conectada.
CAMRelease ()	Ninguno.	0=.OK. -1=.Error.	Desactiva todos los recursos activados por CAMInit.
CAMGetFrame (imagen)	Imagen en escala de grises	Ninguno.	Coge una imagen de la cámara en escala de grises.
CAMGetColFrame (colorimagen,convertir)	Imagen en color. tamaño: 0= imagen color de 24bit. 1=.imagen en escala de grises de 4bit.	Ninguno.	Lee una imagen en color de la cámara.
CAMSet(Brillo, offset, contraste)	Brillo(0-255). Offset (b/n). Hue(color) (0-255). Contraste (0-255).	Ninguno	Establece los valores de la cámara.
CAMGet(brillo, offsetoHue, contrasteosaturación)	enteros para almacenar brillo, offset (b/n) o hue (color) y contraste	Los actuales brillo, offset y contraste (0-255).	Coge los valores actuales de la c'amara.
CAMMode (modo)	Modo= (N0)AUTOBRIGHTNESS	Ninguno.	Pone la cámara en el modo seleccionado.

Cuadro 3: Funciones para la cámara

Formato Función	*P.Entrada*	*P.Salida*	*Descripción*
MotorHandle MOTORInit (DeviceSemantics semantics)	(semantics) Nombre del motor deseado (ver hdt.h).	Manejador del motor.	Inicializa el motor especificado.
int MOTORRelease (MotorHandle handle)	(handle) Lista de todos los manejadores de los motores a los que se le aplicará la función.	0 = Ok 0x11110000 = manejador incorrecto 0x0000xxxx = el parámetro del manejador en el cual solamente esos dígitos binarios siguen siendo conjunto que están conectados con un TPU-channel.	Detiene el funcionamiento de los motores especificados.
int MOTORDrive (MotorHandle handle,int speed)	(handle) Lista de todos los manejadores de los motores a los que se le aplicará la función. (speed) velocidad del motor en porcentaje. Valores: -100 a 100 (negativo hacia atrás) 0 parar el motor	0 = Ok -1 = Manejador incorrecto	Fija la velocidad de los motores especificados

Cuadro 4: Funciones para los motores

Los movimientos resultantes no serán precisos, ya que con este manejo no se tienen en cuenta factores como la inercia del motor, el rozamiento de las ruedas con el suelo, etc. Por ello el uso del control VW para el manejo de los motores resulta más deseable, ya que al tratarse de un sistema realimentado que utiliza los motores junto con los encoders obtiene un control más preciso de los movimientos.

3.2.2. Motores en lazo cerrado: interfaz de movimiento VW

El robot tiene implementado un interfaz de movimiento VW al cual se puede acceder a través de determinadas funciones. No forma parte del sistema operativo RoBIOS, pero es una librería muy útil, proporciona control realimentado de los motores. El control del movimiento que realiza es mucho mas preciso que el conseguido con la interfaz básica, sin realimentación, de RoBIOS.

El uso de este tipo de control requiere la inicialización del mismo. Para controlar los movimientos del eyebot tenemos dos formas de hacerlo:

- mediante funciones que realizan determinados movimientos especificando, generalmente, la distancia a recorrer y la velocidad a la que hay que hacerlo.
- mediante la función `VWSetSpeed`, la cual fija una velocidad objetivo determinada para el eyebot, tanto lineal como angular. Según estos valores el controlador regula

la potencia que se comanda realmente a los motores en cada instante, variando esta para conseguir uno u otro movimiento.

Además de esto también disponemos de funciones para establecer y obtener la posición del eyebot, para averiguar si alguno de los motores está en funcionamiento o para bloquear las llamadas a funciones de la interfaz a la espera que se ejecute alguna de las funciones de la propia interfaz. Una vez finalizado el uso de las funciones de la interfaz es necesario la liberación de los recursos.

A continuación vamos a describir las funciones y los distintos tipos de datos usados por la interfaz VW:

1. **VWHandle:**
   Este tipo de datos nos proporciona un manejador para poder trabajar con la interfaz VW del eyebot.

   ```
   VWHandle vw; /* Definición del manejador para el movimiento del eyebot.*/
   ```

2. **SpeedType:**
   Es el tipo de datos destinado a almacenar valores sobre la velocidad del eyebot. Tiene dos campos, uno para almacenar la velocidad lineal en metros por segundo, v, y otro para almacenar la velocidad angular en radianes por segundo, w.

   ```
   SpeedType velocidad; /* Definición de la variable.*/
   velocidad.v = 3; /* Velocidad lineal en m/s.*/
   velocidad.w = 2; /* Velocidad angular en rad/s.*/
   ```

3. **PositionType:**
   Es el tipo de datos destinado a almacenar el valor de la posición del eyebot. Dispone de tres campos, en uno de ellos se almacena la posición en el eje x (.x), en otro la posición en el eje y (.y), ambos en metros; y en el tercero se guarda la orientación del eyebot en radianes (.phi).

   ```
   PositionType posicion; /* Definición de la variable.*/
   VWGetPosition (vw, &posicion); /* Obtiene posición del eyebot.*/
   LCDPutFloatS (posicion.x, 0, 3); /* Presenta en LCD posición en x.*/
   LCDPutFloatS (posicion.y, 0, 3); /* Presenta en LCD posición en y.*/
   LCDPutFloatS (posicion.phi, 0, 3); /* Presenta orientación del eyebot.*/
   ```

4. **VWInit:**
   Esta función inicializa un manejador, para poder utilizar el resto de las funciones de la interfaz, y devuelve dicho manejador en el caso de que todo haya ido bien, si se ha producido algún error devuelve 0.

```
vw = VWInit (VW_DRIVE,1); /* Inicializa un manejador VW y lo asigna a 'vw'.*/
```

5. **VWRelease:**
   Es una función que dada una variable de tipo VWHandle detiene su funcionamiento liberando el manejador asociado a dicha variable. Si no se ha producido ningún error devuelve 0, en el caso de haber habido errores o se le haya pasado un manejador incorrecto devuelve -1.

   ```
   VWRelease (vw); /* Libera el manejador previamente asignado a vw.*/
   ```

6. **VWStartControl:**
   Pone en funcionamiento un controlador que regula la energía sumistrada a los motores del eyebot para regular su velocidad. Esta función ha de ser utilizada siempre antes que VWSetSpeed, además también puede ser utilizada con las funciones VWDrive_x con lo que conseguimos un movimiento más uniforme y menos brusco que utilizandolas solas. Como parámetros de entrada hay que pasarle una variable de tipo VWHandle y cuatro números reales que indican las componentes proporcionales e integrales tanto de la velocidad lineal como de la velocidad angular. Y como salida devuelve 0 si no ha habido ningún error y -1 en el caso de manejador incorrecto.

   ```
   /* Definimos como constantes las diferentes componentes.*/
   #define V_lin 7.0 /* Componente proporcional de v.*/
   #define T_lin 0.3 /* Componente integral de v.*/
   #define V_ang 10.0 /* Componente proporcional de w.*/
   #define T_ang 0.1 /* Componente integral de w.*/

   /* Llamada a la función.*/
   VWStartControl (vw, V_lin, T_lin, V_ang, T_ang); /* Habilita el controlador.*/
   ```

7. **VWStopControl:**
   Detiene el funcionamiento del controlador que regula la energía suministrada a los motores. Como parámetro de entrada recibe una variable de tipo VWHandle y como salida devuelve 0 si todo ha ido bien y -1 en el caso de que el manejador de entrada sea incorrecto.

   ```
   VWStopControl (vw); /* Para el controlador de energía.*/
   ```

8. **VWSetSpeed:**
   Establece la velocidad de un manejador VWHandle del eyebot, con lo que se consigue que este realice un movimiento determinado; siempre y cuando el controlador de energía esté en funcionamiento. Como parámetros de entrada recibe una variable de

tipo VWHandle, y dos números reales, uno indica la velocidad lineal y el otro la angular. La velocidad lineal se proporciona en metros por segundo, valores positivos indican movimiento hacia adelante y negativos hacia atrás. La velocidad angular se ofrece en radianes por segundo, valores positivos de la misma indican giro hacia la izquierda y negativos giro hacia la derecha. Y devuelve como parámetro de salida 0 en el caso que no se haya producido ningún error y -1 si el manejador proporcionado es incorrecto.

```
/* Damos valor a la variable 'velocidad' definida previamente.*/
velocidad.v = 0.2; /* Velocidad lineal.*/
velocidad.w = 0.2; /* Velocidad angular.*/

VWSetSpeed (vw, velocidad.v, velocidad.w);
/* Eyebot avanza en curva hacia la izq.*/
```

Cuando se utiliza solamente esta función para mover al robot (sin usar `VWDriveStraight` o similares) no debe llamarse a `VWDriveWait` para controlar el tiempo que se mueve el robot con las velocidades comandadas con `VWSetSpeed`. No debe utilizarse `VWDriveWait` porque no evita que la ejecución de `VWSetSpeed` sea interrumpida antes de su finalización. Para controlar bien ese tiempo hay que utilizar la función `OSWait(Time)` pasándole el tiempo que queremos que se mueva con las velocidades comandadas con `VWSetSpeed`.

9. **VWGetSpeed:**
Obtiene la velocidad actual a la que se está desplazando el eyebot. Como parámetros de entrada hay que proporcionarle una variable de tipo VWHandle y un puntero a una variable de tipo SpeedType. Y como salida devuelve en la variable de tipo SpeedType, a la que apunta el puntero dado como argumento de entrada, la velocidad del eyebot tanto lineal, en .v, como angular, en .w, y además devuelve 0 si no se a producido ningún error y -1 en caso contrario.

```
VWGetSpeed (vw, &velocidad); /* Guarda la velocidad del eyebot en 'velocidad'.*/
```

10. **VWSetPosition:**
Establece la posición del eyebot. Como entrada hay que proporcionarle cuatro argumentos, una variable de tipo VWHandle y tres números reales, uno indica la posición en metros sobre el eje x, otro la posición en metros sobre el eje y, y el último la orientación en radianes del eyebot. Como salida devuelve 0 si no ha habido errores y -1 si se ha producido alguno. Para establecer la posición podemos servirnos de los campos de una variable de tipo PositionType ya que puede resultar más claro para el programador. Una vez establecida la posición el eyebot lleba un control de la misma al realizar diferentes movimientos tomando como posición origen la fijada.

```
/* Damos valor a la variable 'posicion' definida previamente.*/
posicion.x = 0.0; /* Posición en el eje x.*/
posicion.y = 0.0; /* Posición en el eje y.*/
posicion.phi = 0.0; /* Orientación del eyebot.*/
/* Llamamos a la función.*/
VWSetPosition (vw, posicion.x, posicion.y, posicion.phi);
```

11. **VWGetPosition:**
Obtine la posición actual del eyebot. Se le pasa como entrada una variable de tipo VWHandle y un puntero a una variable de tipo PositionType. Y devuelve en la variable apuntada por el puntero la posición actual del eyebot, es decir su posición en el eje x (.x) en metros, en el eje y (.y) en metros, además de su orientación (.phi) en radianes.

```
VWGetPosition (vw, &posicion); /* Almacena la posición del eyebot en 'posición'.*/
```

12. **VWDriveStraight:**
Hace avanzar el eyebot una distancia a cierta velocidad. Como entrada tenemos que darle una variable del tipo VWHandle, la distancia a recorrer en metros, en el caso de darle una distancia negativa el eyebot avanzara hacia atras, y la velocidad lineal en metros por segundo, que ha de ser siempre positiva. Hay que tener cuidado ya que como velocidad no podriamos pasarle una variable de tipo SpeedType, aunque si su campo de velocidad lineal (.v). Como salida obtendremos 0 si la función concluye correctamente y -1 si se produce algún error.

```
distancia = 5; /* Distanciaen metros a recorrer.*/
vel = 4; /* Velocidad en metros por segundo.*/
/* Llamada a la función.*/
VWDriveStraight (vw, distancia, vel);
```

13. **VWDriveTurn:**
Gira el eyebot un ángulo determinado. Hay que darle como argumentos de entrada, una variable de tipo VWHandle, el ángulo en radianes que queremos que gire, si es positivo girará hacia la izquierda y si es negativo lo hará hacia la derecha, y la velocidad angular a la que queremos que lo haga, en radianes por segundo. Igual que en el caso anterior como velocidad podríamos utilizar el campo '.w' de una variable de tipo SpeedType. En el caso que la función se termine de ejecutar correctamente devolverá 0 y -1 en caso contrario.

```
/* Gira el eyebot Pi radianes a 1rad/s.*/
VWDriveTurn (vw, (Pi), 1);
```

14. **VWDriveCurve:**
Consigue que el eyebot avance en curva. Como parámetros de entrada hay que proporcionarle una variable de tipo VWHandle, la longitud de la curva a recorrer en metros, el ángulo de giro en radianes, y la velocidad en metros por segundo. Como parámetro de salida obtenemos 0 si todo ha ido bien o -1 si se ha producido algún error.

```
/* Avanza el eyebot 1 metro con un ángulo de (Pi/4)rad. a 0.5m/s.*/
VWDriveCurve (driver, 1, (Pi/4), 0.5);
```

15. **VWDriveRemain:**
Esta función le pasamos como entrada una variable del tipo VWHandle y nos devuelve un real que representa la distancia que queda por recorrer al eyebot tras la ejecución de la última función VWDriveStraight o VWDriveCurve. En el caso de que la ejecución de la última función haya finalizado nos devuelve '0.0'.

```
distancia: float; /* Define la variable.*/
distancia = VWDriveRemain (vw); /* Asigna la distancia por recorrer.*/
```

16. **VWDriveDone:**
Esta función chequea si la última función VWDriveStraight, -Turn, -Curve, ejecutada ha sido completada. Le pasamos como entrada una variable de tipo VWHandle y nos devuelve -1 si se ha producido algún error, 0 si el eyebot se encuentra todavía en movimiento o 1 si la instrucción ha sido completada.

```
/* Llamada a la función.*/
VWDriveDone (vw);
```

17. **VWDriveWait:**
Bloquea las llamadas a procesos a la espera que el último comando VWDriveStright, -Turn, -Curve, se ejecute. Como entrada le pasamos una variable de tipo VWHandle, y como salida obtenemos 0 si el comando VWDrive se ha ejecutado y -1 en caso de que el manejador sea incorrecto o se haya producido algún error.

```
/* Llamada a la función.*/
VWDriveWait (vw);
```

18. **VWStalled:**
Esta función chequea los motores del eyebot para comprobar si por lo menos uno de ellos está parado. Como argumento de entrada hay que proporcinarle una variable de tipo VWHandle y como salida obtenemos -1 si el manejador es incorrecto, 0 si el eyebot está todavía en movimiento y 1 si almenos uno de los motores está parado.

```
/* Llamada a la función.*/
VWStalled (vw);
```

A la hora de programar hemos de tener en cuenta que cada llamada a una función VWDriveStraigth, -Turn, -Curve o VWSetSpeed, acaba con la ejecución de una posible llamada anterior a una a una función VWDriveStraigth, -Turn o -Curve; por lo que tras una llamada a una de las funciones nombradas anteriormente hemos de pensar en la conveniencia de introducir una llamada a VWDriveWait seguidamente o en algo similar para evitar que la ejecución de las funciones sea interrumpida antes de su finalización.

La función `VWDriveStraight` (en conjunción con `VWSetSpeed`) permite hacer que el eyebot avance cierta distancia, pero sólo distancias mayores o iguales a un metro. Para que el eyebot se mueva menos de un metro podemos hacer un programa que utilice la función `VWSetSpeed` y los encoders (ver sección 3.1.2, lectura de encoders cuando se utiliza la librería VW). El siguiente programa ejemplo chequea los pulsos que nos devuelven los encoders, los cuales tienen una relación directa con la distancia avanzada. El programa ordena al eyebot que avance hasta que el valor de los pulsos que nos devuelven sea el adecuado, incluyendo distancias inferiores a 1 metro.

```
#include <eyebot.h>

VWHandle vw; /* Definición del manejador para el movimiento del eyebot.*/

/* constantes.*/
#define V_lin 5    /* Componente proporcional de v.*/
#define T_lin 0.1  /* Componente integral de v.*/
#define V_ang 5    /* Componente proporcional de w.*/
#define T_ang 0.1  /* Componente integral de w.*/
#define encoder_der 0x0304
#define encoder_izq 0x0c02
#define pulsos_metro 3240


float vel , distancia=0.25;

int main ()
{

  vw = VWInit (VW_DRIVE,1);
/* Inicializa un manejador VW y lo asigna a'vw'.*/
  VWStartControl (vw, V_lin, T_lin, V_ang, T_ang);
/* Habilita el controlador.*/
```

Formato Función	*P.Entrada*	*P.Salida*	*Descripción*
VWHandle VWInit (DeviceSemantics semantics, int Timescale)	(semantics) Nombre del V-Omega Driving (Timescale) Escala de actualización (1 to ..)	Manejador del V-Omega Driving 0 para error	Inicializa el VW-Driver (solamente se puede inicializar 1). Los motores y los encoders son reservados automáticamente. La escala de tiempo puede ser escala=1 actualización a 100Hz o escala¿1 actualización a 100/escala Hz.
int VWRelease (VWHandle handle)	(handle) Manejador VW-Driver.	0=ok -1= manejador incorrecto	Detiene el funcionamiento del VW-Driver y para los motores.
int VWSetSpeed (VWHandle handle, meterPerSec v, radPerSec w)	(handle) Manejador VW-Driver (v) velocidad lineal (w) velocidad de rotación	0=ok -1= manejador incorrecto	Fija la velocidad lineal(m/s) y angular(rad/s) del robot.
int VWGetSpeed (VWHandle handle, SpeedType* vw)	(handle) Manejador VW-Driver (vw) Puntero a una estructura que almacena los valores actuales de la velocidad lineal y angular	0=ok -1= manejador incorrecto	Devuelve la velocidad actual del robot, lineal(m/s) y angular(rad/s).
int VWSetPosition (VWHandle handle, meter x, meter y, radians phi)	(handle) Manejador VW-Driver (x) posición sobre el eje x en metros (y) posición sobre el eje y en metros (phi) Orientación en radianes	0=ok -1= manejador incorrecto	Fija la posición del robot.
int VWGetPosition(VWHandle handle, PositionType* pos)	(handle) Manejador VW-Driver (pos) Puntero a una estructura que almacena la posición actual del robot.	0=ok	Devuelve la posición actual del robot.
int VWStartControl(VWHandle handle, float Vv, float Tv, float Vw, float Tw)	(handle) Manejador VW-Driver (Vv) Parámetro para la componente proporcional del controlador de velocidad lineal (Tv) Parámetro para la componente integral del controlador de velocidad lineal (Vw) Parámetro para la componente proporcional del controlador de velocidad angular (Tw) Parámetro para la componente integral del controlador de velocidad angular	0=ok -1= manejador incorrecto	Pone en funcionamiento un controlador (PI-controller) que regula la energía que suministra a los motores para mantener la velocidad fijada (con VWSetSpeed) estable.
int VWStopControl(VWHandle handle)	(handle) Manejador VW-Driver.	0 = ok -1= manejador incorrecto	Deshabilita el controlador (PI-Controller).
int VWDriveStraight (VWHandle handle, meter delta, meterpersec v)	(handle) Manejador VW-Driver (delta) distancia a conducir en m (positiva adelante) (negativa atrás) (v) velocidad (siempre positiva).	0 = ok -1= manejador incorrecto	Avanza o retrocede el robot el número de metros "deltaçon velocidad "v". Cualquier llamada a VWDriveStraight, -Turn, -Curve or VWSetSpeed interrumpirá la ejecución de este comando.

Cuadro 5: Funciones del interfaz VW

Formato Función	*P.Entrada*	*P.Salida*	*Descripción*
int VWDriveTurn (VWHandle handle, radians delta, radPerSec w)	(handle) Manejador VWDriver (delta) ángulo de giro en radianes (negativo dirección de las agujas del reloj). (w) velocidad de giro (siempre positiva)	0=ok -1= manejador incorrecto	Hace girar el robot un ángulo "delta"en radianes con una velocidad "w". Cualquier llamada a VWDriveStraight, -Turn, -Curve or VWSetSpeed interrumpirá la ejecución de este comando.
int VWDriveCurve (VWHandle handle, meter delta_l, radians delta_phi, meterpersec v)	(handle) Manejador VWDriver (delta_l) longitud del segmento de la curva en metros (delta_phi) ángulo de giro en radianes (negativo dirección de las agujas del reloj) (v) velocidad (siempre positiva)	0=ok -1= manejador incorrecto	Conduce el robot en curva de longitud "delta_l", ángulo de giro "delta_phiz velocidad "v". Cualquier llamada a VWDriveStraight, -Turn, -Curve or VWSetSpeed interrumpirá la ejecución de este comando.
float VWDriveRemain (VWHandle handle)	(handle) Manejador VWDriver	0.0 = si el comando VWDrive previo ha sido completado. Cualquier otro valor = distancia que le queda por recorrer.	Devuelve la distancia que le falta por recorrer fijadas mediante VWDriveStraight, -Turn (para -Curve sólo devuelve "delta_l")
int VWDriveDone (VWHandle handle)	(handle) Manejador VWDriver.	-1= Manejador incorrecto 0 = El vehículo está todavía en movimiento. 1 = El comando previo VWDrivex ha sido completado.	Chequea si el comando previo VWDrivex ha sido completado
int VWDriveWait (VWHandle handle)	(handle) Manejador VWDriver.	-1= Manejador incorrecto. 0 = El comando previo VWDrivex ha sido completado.	Bloquea la llamada a procesos hasta que el comando previo VWDrivex haya sido completado
int VWStalled (VWHandle handle)	(handle) Manejador VWDriver.	-1= Manejador incorrecto. 0 = El robot está todavía en movimiento o el comando que bloquea el movimiento está activo. 1 = Al menos uno de los motores del robot está parado durante el comando VW driving.	Chequea si al menos uno de los motores del robot está parado.

Cuadro 6: Funciones del interfaz VW (II)

```
  sale=0;
  pulsos = fabs((int)(pulsos_metro * distancia));

  do{
       VWSetSpeed(vw, vel,0);
/*Ordenamos al eyebot que avance en linea recta */

  }while ((fabs(QUADRead(encoder_der))<= pulsos) &&
(fabs(QUADRead(encoder_izq)) <= pulsos));

  VWSetSpeed(vw, 0,0); /*Ordenamos al eyebot se pare */

  VWRelease (vw); /* Libera el manejador previamente asignado a vw.*/
  VWStopControl (vw); /* Para el controlador de energía.*/

  return 1;
}
```

Al utilizar la función `VWSetSpeed` la velocidad será controlada directamente por nuestro código y no por la librería VW. Por esto hay que ser cuidadoso para no quemar los motores o desvirtuar el movimiento rectilíneo: la velocidad de desplazamiento de eyebot no podrá ser excesivamente grande, sobre todo si la distancia es menor a un metro. Para menos de un metro la velocidad adecuada es 0.4 m/s.

En la práctica se hemos encontrado dificultades para hacer que el Eyebot gire sobre sí mismo (con un ángulo muy cerrado), parece que en esta situación le falta fuerza a los motores. El problema se debe a que el eyebot apoya todo el peso de la batería y el LCD sobre el pivote trasero de plástico. Como en estos giros utiliza sólo uno de los motores la fuerza de éste es inferior a la fuerza de rozamiento con el suelo, entonces el eyebot no puede girar o gira con dificultad.

3.2.3. Acceso a los servos

El uso de los servos es muy sencillo. Al igual que la mayoría de los elementos hardware hay que inicializarlos, para ello se utiliza la función SERVOInit (nombre_servo_en_HDT). Para fijar el servo a un ángulo deseado se necesita la función SERVOSet(servo,ángulo). Es conveniente liberar el recurso (SERVORelease (servo)) tras finalizar su uso.

Formato Función	*P.Entrada*	*P.Salida*	*Descripción*
ServoHandle SERVOInit(DeviceSemantics semantics)	(semantics) Nombre del servo deseado (ver hdt.h)	Manejador del servo	Inicializa el servo especificado
int SERVOSet (ServoHandle handle,int angle)	(handle) Lista de todos los manejadores de los Servos a los que se le aplicará la función (angle) Ángulo del servo Valores:0-255	0 = Ok -1 = error manejador incorrecto	Fija los servos seleccionados al ángulo introducido
int SERVORelease (ServoHandle handle)	(handle) Lista de todos los manejadores de los Servos a los que se le aplicará la función	0= Ok 0x11110000 = manejador incorrecto 0x0000xxxx = el parámetro del manejador en el cual solamente esos dígitos binarios siguen siendo conjunto y están conectados con un TPU-channel	Detiene el funcionamiento de los servos especificados

Cuadro 7: Funciones para los servos

3.3. Acceso a los sistemas de comunicaciones

3.3.1. Puerto serie

El programa que está siendo ejecutado en el Eyebot puede comunicarse a través del puerto serie con otra máquina, enviando y recibiendo datos o comandos. Para ello lo primero que se ha de hacer es inicializar el puerto serie a través de la función OSInitRS232(velocidad,handshake,interfaz), indicándole la velocidad de transmisión, el interfaz donde está el puerto serie y si se desea el uso del handshake (visualización de bytes transmitidos).

Las funciones OSSendCharRS232(char,interfaz), OSSendRS232(char,interfaz) se encargan de enviar datos, y la función OSRevRS232(cadena,interfaz) se encarga de la recepción.

Es muy importante destacar que tanto la lectura como el envío de datos solamente puede realizarse de carácter en carácter.

Si se desea hacer una inicialización de los buffers de entrada y de salida es posible hacer una limpieza de ellos utilizando las funciones OSFlushInRS232 (interface) y OSFlushOutRS232 (interface) respectivamente.

3.3.2. Puerto paralelo

La depuración de los programas cargados en el eyebot se realiza a través del puerto paralelo. En él se van cargando los datos de salida con la función OSWriteParData(value), y se leen los de entrada con OSReadParData(). Es posible acceder al registro de control a través de OSReadParCTRL() para su lectura y de OSWriteParCTRL(value) para la

Formato Función	*P.Entrada*	*P.Salida*	*Descripción*
OSDownload (programa, bytes, vel, handshake, interfaz)	Nombre del programa. Bytes a transferir. Velocidad (SER9600, SER19200, SER38400, SER57600, SER115200(sólo SERIAL2-3)). Hanshake (NONE, TRSCTS). Interfaz (SERIAL1-3).	0= no error, descarga incompleta. 99= descarga completa. 1= timeout en la recepción. 2= error en la recepción. 3= timeout en el envío. 4= srec checksum error. 6= cancelación por el usuario. 7= error srecord desconocido. 8= baudrate no válido. 9= dirección de comienzo no válida. 10= interfaz no válido.	Carga un programa en el robot.
OSInitRS232 (vel, handshake, interfaz)	Velocidad de transferencia (SER9600, SER19200, SER38400, SER57600, SER115200). Handshake (NONE, RTSCTS). Interfaz (SERIAL1-3)	0= OK. 8= velocidad no permitida. 10= interfaz no válido.	Inicializa el puerto con protocolo RS232 con las características especificadas.
OSSendCharRS232(char, interface)	Carácter a enviar. Interfaz (SERIAL1-3).	0= OK. 3= timeout en el envío. 10= interfaz no válido.	Manda un carácter por el puerto especificado con el protocolo RS232.
OSSendRS232 (cadena, interface)	Cadena a enviar. Interfaz (SERIAL1-3)	0= OK. 3 = timeout en el envío. 10= interfaz no válido	Manda una cadena de caracteres usando la función anterior.
OSRecvRS232 (cadena, interface)	Carácter. Interfaz (SERIAL1-3).	0= OK. 1= timeout en la recepción. 2= error en la recepción. 10= interfaz ilegal.	Recibe una cadena por el puerto especificado.
OSCheckOutRS232 (interface)	Interfaz (SERIAL1-3).	¿0 número de caracteres esperando en FIFO. ¡0 0xffffff0a= interfaz ilegal.	Devuelve el número de caracteres que actualmente espera en la FIFO.
OSCheckInRS232 (interface)	interfaz (SERIAL1-3)	¿0 número de caracteres disponibles en FIFO. ¡0 0xffffff02= error de recepción, (no hay caracteres disponibles). 0xffffff0a= interfaz ilegal.	Devuelve el número de caracteres que actualmente acepta la FIFO.
OSFlushInRS232 (interface)	interfaz (SERIAL1-3)	0= OK, 10= interfaz ilegal	Resetea el estado del receptor y limpia su pila(FIFO). Muy usado en modo NOHANDSHAKE para inicializar antes de empezar a recibir.
OSFlushOutRS232 (interface)	Interfaz (SERIAL1-3)	0= OK. 10= interfaz ilegal.	Limpia la pila del transmisor. Muy usado para abortar la emisión actual al host por ejemplo cuando éste no responde.

Cuadro 8: Funciones para acceder al puerto serie

Formato Función	*P.Entrada*	*P.Salida*	*Descripción*
BYTE OSReadParData (void)	Ninguna	Estado actual del los 8bits de datos del puerto paralelo	Lee el contendido del puerto de paralelo.
void OSWriteParData (BYTE value)	(value) Nuevo dato de entrada.	Ninguno.	Escribe un nuevo dato de salida en el puerto paralelo.
BYTE OSReadParSR (void)	Ninguno	Estado actual del registro de estado del puerto paralelo (5 bits).	Lee el estado actual del registro de estado del puerto paralelo, activos a nivel alto (BUSY(4), ACK(3), PE(2), SLCT(1), ERROR(0)).
void OSWriteParCTRL (BYTE value)	(value) Nuevo valor para el registro de control del puerto paralelo (4bits).	Ninguno.	Escribe un nuevo valor en el registro de control, activo a nivel alto (SLCTIN(3), INT(2), AUTOFDXT(1), STROBE(0))
BYTE OSReadParCTRL (void)	Ninguno.	Estado actual del registro de control.	Lee el valor actual del registro de control (SLCTIN(3), INT(2), AUTOFDXT(1), STROBE(0))

Cuadro 9: Funciones para el puerto paralelo

escritura en él.

3.3.3. Radiocomunicaciones

El uso de la radiocomunicación requiere una inicialización previa a través de RADIOInit(). Después de esta inicialización es posible tanto la recepción de mensajes con la función RADIORecv(robot,longitud,mensaje) como el envío de mensajes (individuales o en multidifusión) con la función RADIOSend(robot,longitud,mensaje).La función RADIOCheck() permite verificar si hay un mensaje a la espera de ser recibido. En la finalización del programa es necesario cerrar la radiocomunicación con RADIOTerm().

A continuación se van a describir con detalle todas las funciones y los tipos usados en la comunicación por radio entre los eyebots:

1. **RadioInit:**
   Inicializa y pone en marcha la radio comunicación.
   Sintáxis: int RADIOInit()

   La función devuelve un 0 si todo se ha inicializado correctamente.

2. **RadioTerm:**
   Finaliza la radio comunicación.
   Sintáxis: int RADIOTerm()

La función devuelve un 0 si se ha finalizado correctamente.

3. **RADIOSend:**
Envía un mensaje a otro robot. El mensaje es enviado en segundo plano, la longitud del mensaje debe ser menor o igual a la constante MAXMSGLEN. Esta constante está declarada en directorioeyebot/include/types.h y su valor por defecto es 35, por tanto, el número máximo de bytes a enviar en un solo mensaje no puede superar los 35 bytes.
Sintáxis: int RADIOSend(BYTE id, int longitud, BYTE* buffer)

La función recibe tres parámetros de entrada:

a) id: Identidicador del eyebot destino. Este ID debe ser un valor comprendido entre 1 y MAXEYE (esta constante está declarada en directorioeyebot/include/types.h y su valor por defecto es 16, número máximo de eyebots comunicandose a la vez).

b) longitud: Número de bytes enviados en el mensaje. Este valor debe estar comprendido entre 1 y MAXMSGLEN.

c) buffer: Puntero a una variable de tipo BYTE. Esta variable normalmente será un array de BYTE, donde se almacenará en cada posición cada carácter o entero del mensaje.

La función devuelve un 0 si la transmisión ha ído bien y un 1 si el buffer está lleno o el mensaje es demasiado largo.

¡OJO!: Hay que tener cuidado al hacer varios send seguidos, pues se puede saturar el buffer de emisión. Para solucionar este problema se puede intercalar una instrucción de espera (OSWait(<centesimas >)) entre cada send.

4. **RADIOCheck:**
Función que retorna el número de mensajes almacenados en el buffer. Esta función puede ser llamada antes de recibir. Se puede utilizar para que la función RADIO-Recv no sea bloqueante, utilizandola solo cuando RADIOCheck dice que ya hay algun mensaje que recibir.
Sintáxis: int RADIOCheck()

Ejemplo que recibe y muestra en la pantalla el ID origen, longitud y contenido del msg:

- RADIORecv bloqueante:
```
RADIORecv(&fromId, &len, mes);
LCDPrintf(‘‘Recibido%d-%d:%3d\a\n’’, fromId, len, mes[0]);
```

- RADIORecv no bloqueante:
```
if (RADIOCheck())
RADIORecv(&fromId, &len, mes);
LCDPrintf(‘‘Recibido%d-%d:%3d\a\n’’, fromId, len, mes[0]);
```

5. **RADIORecv:**
   Retorna el siguiente mensaje contenido en el buffer. Los mensajes son devueltos en el orden en que se recibieron. Recibir bloqueará el proceso que la ha invocado si no hay ningún mensaje en el buffer, hasta que llegue el siguiente mensaje.
   Sintáxis: int RADIORecv(BYTE* id, int* longitud, BYTE* buffer)

   La función devuelve tres parámetros de salida:

   a) id: Puntero a un BYTE donde se va a almacenar el identificador (ID) del robot emisor del mensaje.

   b) longitud: Puntero a un entero donde se guardará la longitud del mensaje recibido,

   c) buffer: Puntero a un BYTE (normalmente será un array de Bytes) donde se almacenará el mensaje recibido. Se modificarán tantas posiciones de buffer como indique longitud.

6. **RADIOGetStatus:**
   Retorna la información del estado actual de la radio comunicación (el identificador de eyebot y la lista de eyebots activos en ese momento en la red).
   Sintáxis: int RadioGetStatus(RadioStatus *s)

   La función devuelve un puntero a un registro de tipo RadioStatus definido en directorioeyebot/include/types.h. El registro posee dos campos: El campo master (de tipo BYTE) almacena el identificador del eyebot y el campo active (de tipo array de BOOL) almacena en el array si los robots están activos en la red (1 = activo, 0 = inactivo).

También es posible comunicar uno o varios eyebots con un PC. El emisor/receptor se conecta al puerto serie (COM2) y al teclado. La comunicación es bastante inestable. Algunos trucos que ayudan son arrancar los programas a ejecutar en los eyebots antes de hacerlo en el PC y reiniciar la sesión del PC si se desconecta la radio.

Las funciones que pueden utilizarse en el PC son las anteriormente descritas entre eyebots así como RADIOGetIoctl para leer la configuración de los parámetros de radio y RADIOSetIoctl para cambiar la configuración de dichos parámetros.

1. **RADIOGetIoctl:**
   Lee la configuración de los parámetros de radio (interface, velocidad, id, remoteOn, imageTransfer y debug). Estos parámetros forman parte de un registro llamado RadioIOParameters que se puede encontrar en directorioeyebot/include/types.h.
   Sintáxis: int RADIOGetIoctl(RadioIOParameters* radioParams)

   La función devuelve un parámetro de salida que será el registro radioParams. Ahora accediendo a los campos de este registro podemos comprobar la configuración de todos los parámetros (ver types.h). Por ejemplo, para comprobar la velocidad de la transmisión haríamos:

```
RadioIOParameters radioParams;
main()
RADIOGetIoctl(&radioParams);
LCDPutInt(radioParams.speed);
...
```

2. **RADIOSetIoctl:**
   Cambia la configuración de los parámetros de radio. Esta función debe ser llamada antes de la llamada a RADIOInit(). Utiliza el mismo parámetro que la función anterior pero ahora es de entrada.
   Sintáxis: int RADIOSetIoctl(RadioIOParameters radioParams)

   La manera de modificar los parámetros es modificar cada campo del registro radioParams y luego llamar a la función. Por ejemplo:

```
RadioIOParameters radioParams;
main()
radioParams.speed = SER38400;
radioParams.interface = SERIAL3; /* COM 3 */
RADIOSetIoctl(radioParams);
...
```

Ejemplo de programa que autodetecta los robots presentes en el radio de acción de cada robot y muestra en pantalla el identificador de cada nodo detectado:

```
#include "eyebot.h"
int main()
{
```

Formato Función	*P.Entrada*	*P.Salida*	*Descripción*
int RADIOInit (void)	Ninguno.	0 =Ok	Inicializa y pone en marcha la radio comunicación.
int RADIOTerm (void)	Ninguno.	0 =Ok	Finaliza la radio comunicación.
int RADIOSend (BYTE id, int byteCount, BYTE* buffer)	(id) Número Identificador el robot destinatario del mensaje. Es posible hacer un BROADCAST multidifusión. (byteCount) Longitud del mensaje. (buffer) Contenido del mensaje.	0 = OK 1 = Si el buffer está lleno o el mensaje es demasiado largo	Envía un mensaje a otro robot. El mensaje es enviado en segundo plano, la longitud del mensaje debe de ser menor o igual a MAXMSGLEN. Los mensajes pueden ser enviados en BROADCAST.
int RADIOCheck (void)	Ninguno.	Retorna el número de mensajes almacenados en el buffer.	Función que retorna el número de mensajes almacenados en el buffer. Esta función puede ser llamada antes de recibir.
int RADIORecv (BYTE* id, int* bytesReceived, BYTE* buffer)	Ninguno.	(id) Identificador del Robot que envía el mensaje. (bytesRecevive) Longitud del mensaje. (buffer) Contenido del mensaje	Retorna el siguiente mensaje contenido en el buffer. Los mensajes son devueltos en el orden en el que se recibieron. Recibir bloqueará la llamada a procesos si no hay ningún mensaje en el buffer hasta que llegue el siguiente mensaje..
void RADIOGetIoctl (RadioIOParameters* radioParams)	Ninguno	(radioParams) Configuración de parámetros de radio.	Lee la configuración de los parámetros de radio.
void RADIOSetIoctl (RadioIOParameters* radioParams)	(radioParams) Nueva configuración para los parámetros de radio.	Ninguno.	Cambia la configuración de los parámetros de radio. Esta función debe ser llamada antes de la llamada a RADIOInit().
int RADIOGetStatus (RadioStatus *status)	Ninguno.	(status) Estado actual de la radio comunicación.	Retorna la información del estado actual de la Radio comunicación.

Cuadro 10: Funciones para la radiocomunicación

```
  BYTE myId;
  RadioStatus status;
  int i, err;

  myId = OSMachineID();
  err = RADIOInit();
  do
  {
        LCDPutString("Detectando     -\n");
        LCDLine(0,10,127,10,1);
        LCDMenu(" "," "," ","END");

        RADIOGetStatus(&status);
        for(i = 1;i <= 16; i++)
                {
                if ((status.active[i]) && (i != myId))
                LCDPrintf("%d ",i);
                }
        LCDSetPos(0,15);
        OSWait(50);
        LCDPrintf("|\n");
        OSWait(50);
        LCDClear();
  }while (KEYRead() != KEY4);

  RADIOTerm();
  return 0;
}
```

Ejemplo de programas de comunicación entre un PC y dos eyebots:

eye_radio.c: Se ejecuta en dos eyebots de tal forma que cada uno envía un caracter a un PC (en su turno) y espera a recibir antes de enviar el siguiente. Los robots deben tener IDs 1 y 2 respectivamente. Finaliza pulsando la tecla FIN. La secuencia de envíos será robot 1, PC, robot 2, PC, robot 1.....

```
#include "eyebot.h"

int main()
{ BYTE myId, nextId, fromId;
  BYTE mes[50]; /* buffer para los mensajes */
  int  len, err;
```

```
  LCDMenu(" "," "," "," ","END");

  myId = OSMachineID();
  if (myId==0) { LCDPutString("Error RadioLib\n"); return 1; }
    else LCDPrintf("Mi ID es  %d\n", myId);
  switch(myId)
  { case 1 : nextId = 0; break;
    case 2 : nextId = 0; break;
    default: LCDPutString("Set ID to 1 o 2\n"); return 1;
  }

  LCDPutString("Radio\n");
  err = RADIOInit();
  if (err) { LCDPutString("Error Radio Init\n"); return 1; }
    else LCDPutString("Init\n");

  if (myId == 1)  /* robot 1 realiza el primer envío */
  { mes[0] = 0;
    err = RADIOSend(nextId, 1, mes);
    if (err) { LCDPutString("Error Send\n"); return 1; }
    else LCDPrintf ("PRIMER ENVIO\n");
  }

  while ((KEYRead()) != KEY4)
  { if (RADIOCheck())  /* si hay un mensaje esperando */
    { RADIORecv(&fromId, &len, mes);  /* recibe el mensaje */
      LCDPrintf("Recv %d-%d: %3d\a\n", fromId, len, mes[0]);
      mes[0]++;              /* incrementa el número y lo envía */
      err = RADIOSend(nextId, 1, mes);
      LCDPrintf("Enviando\n");
      if (err) { LCDPutString("Error Send\n"); return 1; }
    }
  }
  RADIOTerm();
  return 0;
}
```

pc_radio.c: Espera a recibir un caracter de un eyebot con ID 1 antes de enviar un caracter a un robot con ID 2. A continuación espera a recibir un caracter de un eyebot con ID 2 antes de enviar un caracter a un robot con ID 1 y así sucesivamente. 15 mensajes

enviados.

```
#include "remote.h"
#include "eyebot.h"
#include "stdio.h"

int main()
{ BYTE myId, nextId, fromId;
  BYTE mes[50]; /* buffer para los mensajes */
  int  len, err, i;
  RadioIOParameters radioParams;

  RADIOGetIoctl(&radioParams);
  radioParams.speed = SER38400;
  radioParams.interface = SERIAL2;
  RADIOSetIoctl(radioParams);

  printf("Radio\n");
  err = RADIOInit();
  if (err)
    {
     printf("Error Radio Init\n");
     return 1;
    }
  else
     printf("Init\n");

  /* PC (ID 0) espera a recibir un caracter del eyebot 1 antes de enviar al 2 */
  nextId = 2;

  /* Realiza 15 envíos */
  i = 0;

  while (i < 15)
  {
   if (RADIOCheck())  /* comprueba si hay un mensaje esperando */
    { RADIORecv(&fromId, &len, mes);  /* recibe el siguiente mensaje */
      printf("Recv %d-%d: %3d\a\n", fromId, len, mes[0]);
      mes[0]++;              /* incrementa el número y envía mensaje */
      err = RADIOSend(nextId, 1, mes);
      i++;
```

```
      if (err)
        {
         printf("Error Send\n");
         return 1;
        }
      if (nextId == 2)
        nextId = 1;
      else
        nextId = 2;
    }
  }

  printf("15 envíos realizados\n");
  RADIOTerm();
  return 0;
}
```

El programa a ejecutar en el PC debe incluir ¡remote.h¿en el código fuente. Para su compilación es necesario incluir la librería de radio libradio.a en /usr/local/eyebot/mc/pc-appl/wireless/linux/lib.

Ejemplo de makefile para compilar los dos programas anteriores:

```
all: compila_pc compila_eyebot descarga

compila_pc: pc_radio.c
        gcc pc_radio.c  -I/usr/local/eyebot/mc/include \
                        -I/usr/local/eyebot/gcc-m68k/include \
                        -L. \
                        -lradio
        mv a.out pc_radio.hex

compila_eyebot: eye_radio.c
        gcc68 eye_radio.c

descarga: eye_radio.hex
        dl eye_radio.hex

clean:
        rm -rf *.hex *.o

version:
```

```
tar -cvzf radio_pc_eyebot--tgz *.c makefile *.hex *.h *.a README
```

3.4. Recursos de multiprogramación

3.4.1. Tareas

Para el Eyebot es posible la multiprogramación de dos maneras distintas, con desalojo y sin desalojo, también llamado método cooperativo.

En el modo **cooperativo** sólo se ejecuta aquella tarea que tiene el testigo, de manera que si se bloquea ninguna otra tarea podrá ejecutarse porque el control de flujo está controlado por la tarea bloqueada.

Por el contrario, si las tareas están ejecutándose con **desalojo**, el control de flujo es independiente de las propias tareas. Periódicamente el sistema operativo desaloja a la tarea actual y le asigna el uso del procesador a otra, de manera que si una tarea se detiene las restantes podrán seguir ejecutándose.

Para trabajar en modo multitarea con el Eyebot es necesario elegir el método de multiprogramación que se necesite, siendo el que más se ajusta a nuestras necesidades el método con desalojo por lo anteriormente comentado.

A continuación, veremos una breve descripción de las funciones más importantes que aparecen en la tabla inferior:

1. **OSMTInit (modo) :**
   Inicializa el entorno multitarea en el modo pasado. El argumento `modo` puede, por tanto, tomar dos valores posibles, COOP y PREEMPT, que indican modo cooperativo (paso de testigo) y modo no cooperativo (con desalojo), respectivamente.

2. **OSMTStatus ( ) :**
   Devuelve el actual modo de la multitarea: COOP, PREEMPT o NOTASK. Devolverá NOTASK si no se ha inicializado previamente el entorno multitarea (véase `OSMTInit`).

3. **tcb* OSSpawn (NombreTarea, DirComienzo, TamPila, Prioridad, uid) :**
   Inicializa una tarea con los argumentos pasados y la inserta en el planificador pero sin ponerla a lista para ejecutar.

   ```
   ...
   struct tcb* tarea1; /* puntero al bloque de control */
   ...
   ```

```
void funcion1(){...}
...
int main ()
{
...
OSMTInit(PREEMPT); /* Por poner un ejemplo, usamos modo con desalojo */

/* vamos a inicializar la tarea1 */
tarea1 = OSSpawn("Un_nombre_cualquiera", funcion1, 8510, MIN_PRI, 0);
...
}
```

4. **OSReady (tcb* thread) :**
   Una vez que se ha llamado a `OSSpawn` para inicializar la tarea en cuestión, llamando a esta función se coloca la tarea en la cola de procesos listos para ejecutar.
   Si esta función es llamada desde alguna tarea cuando el planificador ya está en funcionamiento, es decir, cuando la función `OSPermit` ya a sido llamada para pasar de monotarea a multitarea, la tarea pasada como argumento se pondrá a listo.

```
...
struct tcb* tarea1; /* puntero al bloque de control */
...
void funcion1(){...}
...
int main ()
{
...
OSMTInit(PREEMPT); /* Por poner un ejemplo, usamos modo con desalojo */

/* vamos a inicializar la tarea1 */
tarea1 = OSSpawn("Un_nombre_cualquiera", funcion1, 8510, MIN_PRI, 0);
...
/* ponemos "tarea1" a listo */
OSReady (tarea1);
...
}
```

5. **OSPermit ( ) :**
   Usaremos esta función sólo cuando trabajemos en modo con desalojo.
   En el momento en que llamamos a `OSPermit` desde alguna función en monotarea, comienzan a ejecutarse las tareas listas (ver `OSReady`), quedando bloqueada la monotarea en el punto desde el que se llamó a `OSPermit` hasta que la cola de procesos listos

de la multitarea se vacíe, momento en el cual se retomará la ejecución en monotarea. La manera más frecuente de vaciar la cola de listos es usar una **tarea maestro** que mata a todas las tareas esclavo en algún momento y se suicida luego ella misma. Sin embargo, también es posible hacer que las tareas lanzadas se suiciden en algún momento de sus vidas, **sin jerarquía entre tareas**.
Véase la función `OSKill`.

6. **OSKill (tcb* tarea) :**
Mata la tarea pasada como parámetro y replanifica. Si le pasamos 0, la tarea que llama a esta función se suicida.
Ejemplo de vaciado de la cola de tareas con tarea maestro:

```
...
struct tcb* tarea1, master;
...
void funcion1(){...}
void maestro()
{
...
if (condición)
{
OSKill(tarea1); /* el maestro mata al esclavo */
OSKill(0); /* el maestro se suicida */
}
}
...
int main ()
{
...
OSMTInit(PREEMPT); /* Por poner un ejemplo, usamos modo con desalojo */

/* vamos a inicializar la tarea1 */
tarea1 = OSSpawn("esclavo", funcion1, 8510, MIN_PRI, 0);
master = OSSpawn("maestro", maestro, 8510, MAX_PRI, 1);
...
OSReady (tarea1);
OSReady (master);
OSPermit();
  /* quedamos bloqueados aquí hasta que la tarea master mate a tarea1 y a sí misma */
...
return 0;
}
```

Ejemplo de vaciado de la cola de tareas sin jerarquías entre tareas:

```
...
struct tcb* tarea1, tarea2;
...
void funcion1()
{
...
OSKill(0); /* la tarea se suicida */
...
}
void funcion2()
{
...
OSKill(0); /* la tarea se suicida */
...
}
...
int main ()
{
...
OSMTInit(PREEMPT); /* Por poner un ejemplo, usamos modo con desalojo */

/* vamos a inicializar la tarea1 */
tarea1 = OSSpawn("tarea1", funcion1, 8510, MIN_PRI, 0);
tarea2 = OSSpawn("tarea2", funcion2, 8510, MIN_PRI, 1);
...
OSReady (tarea1);
OSReady (tarea2);
OSPermit();
  /* quedamos bloqueados aquí hasta que las tareas lanzadas se suiciden */
...
return 0;
}
```

7. **OSReschedule ( ) :**
   Lo que hace esta función es elegir una nueva tarea. Cuando trabajemos en modo cooperativo, la usaremos como equivalente de la función `OSPermit` para activar el planificador.
   Al igual que con `OSPermit`, la función en monotarea desde la que se llama a `OSReschedule` quedará bloqueada hasta que la cola de procesos listos de la multitarea se vacíe.

Usada ya en multitarea desde una de las tareas que ejecutan en modo cooperativo, se pasa el testigo a otra tarea.

8. **OSSleep (TiempoCentésimas) :**
   Si hemos inicializado la multitarea, en el modo que sea, y hemos activado ya el planificador, el thread que llame a esta función quedará parado durante el tiempo indicado y, además, pasará el control al siguiente thread listo que haya elegido el planificador. El thread parado no podrá retomar el control hasta que haya pasado el tiempo.
   Si llamamos a `OSSleep` en monotarea, su funcionamiento será idéntico al de la función `OSWait(tiempo)`.

9. **OSSuspend (tcb* thread) :**
   Pone el thread pasado a suspendido.

10. **OSRun (tcb* thread) :**
    Pone a listo el thread pasado y replanifica.
    Un thread suspendido podrá ser puesto de nuevo a listo llamando desde otra tarea que esté activa a `OSRun(thread-suspendido)` o a `OSReady(thread-suspendido)`, la única diferencia es que con `OSRun` se replanificará y con `OSReady` no.

11. **OSYield ( ) :**
    Su función es la misma que `OSSuspend`, solo que pone a suspendido el thread desde el que es llamada (por eso no recibe argumentos) y replanifica.

En la siguiente tabla el parámetro `thread` es un puntero a la estructura tbl o 0 para el thread actual.

Ejemplo de multitarea en modo cooperativo:

```
#define NUM_TAREAS 2   /* número de tareas */
#define SSIZE 8510   /* tamaño de la pila para cada tarea */

struct tcb* tarea_p[SLAVES];   /* array de punteros a bloques de control */

void funcion0 ()
{
   int contador;
   for (contador=0; contador < 1000; contador++)
   {
       printf(''soy la funcion 0\n'');
       OSReschedule(); /* Pasamos el testigo a otra tarea. En lugar de OSReschedule podría
                          usarse OSSleep(tiempo), por ejemplo. */
   }
```

Formato Función	*P.Entrada*	*P.Salida*	*Descripción*
OSMTInit (modo)	Modo de operación. COOP (por defecto). PREEMT	Ninguno.	Inicializa el entorno multitarea
OSSpawn (nombre, dircom, tampil, prioridad, uid)	Nombre de la tarea Dirección de comienzo. Tamaño de su pila. Prioridad(MINPRI-MAXPRI). Identificador	Puntero al bloque de control de la tarea inicializada.	Devuelve el thread inicializado e insertado en el planificador pero no puesto a listo.
OSMTStatus	Ninguno.	Modo multitarea PREEMPT, COOP, NOTASK	Devuelve el modo de la actual multitarea.
OSReady (thread)	Puntero al bloque de control de una tarea.	Ninguno.	Pone el thread a listo.
OSSuspend (thread)	Puntero al bloque de control de una tarea.	Ninguno.	Pone el valor del thread a suspendido.
OSReschedule	Ninguno	Ninguno	Elige una nueva tarea.
OSYield	Ninguno.	Ninguno.	Suspende el actual thread y replanifica.
OSRun (thread)	Puntero al bloque de control de una tarea.	Ninguno.	Pone a listo el thread dado y replanifica.
OSGetUID (thread)	Puntero al bloque de control de una tarea.	UID de la tarea.	Devuelve el identificador del thread dado.
OSKill (thread)	Puntero al bloque de control de una tarea.	Ninguno.	Elimina el thread pasado y replanifica.
OSExit (codigo)	Código de salida.	Ninguno.	Mata el thread actual con el código de salida y mensaje.
OSPanic (mensaje)	Mensaje de texto	Ninguno.	Cuando se produce un error, imprime el mensaje y para el procesador.
OSSleep (tiempo)	Tiempo en centésimas de segundo (1/100).	Ninguno.	Permite al thread pararse durante el tiempo indicado, en multitarea se pasa el control a otro thread, es una llamada a OSWait en monotarea.
OSForbid	Ninguno.	Ninguno.	Desactiva el thread pasando a modo PREEMPT.
OSPermit	Ninguno.	Ninguno.	Activa el thread pasando a modo PREEMPT.

Cuadro 11: Funciones para las tareas y procesos

```
   OSKill(0); /* La tarea se suicida, dejando su lugar en la cola de listos libre */
}

void funcion1 ()
{
   int contador;
   for (contador=0; contador < 1000; contador++)
   {
       printf(''soy la funcion 1\n'');
       OSReschedule(); /* Pasamos el testigo a otra tarea. En lugar de OSReschedule podría
                          usarse OSSleep(tiempo), por ejemplo. */
   }
   OSKill(0); /* La tarea se suicida, dejando su lugar en la cola de listos libre */
}

int main()
{
  /*inicialización de la multitarea en modo cooperativo */
  OSMTInit(COOP);

  /*inicialización de las tareas*/
  tarea_p[0] = OSSpawn("tarea0", funcion0, SSIZE1, MAX_PRI, 1);
  tarea_p[1] = OSSpawn("tarea1", funcion1, SSIZE1, MIN_PRI, 0);

  /*puesta a punto de las tareas*/
  OSReady(tarea_p[0]);
  OSReady(tarea_p[1]);

  /*inicialización del planificador*/
  OSReschedule(); /* En este punto la monotarea retomará el control cuando la cola de listo
                  de la multitarea quede vacía, es decir, cuando se suiciden las dos tareas
  return 0;
}
```

Puesto que en modo cooperativo las tareas han de esperar a que la tarea en ejecución actual pase el testigo, en este ejemplo, al solo haber dos tareas, el resultado siempre será el mismo: las dos tareas escriben su mensaje en pantalla alternadamente.

Ejemplo de multitarea en modo con desalojo:

```
#define NUM_TAREAS 2   /* número de tareas */
```

```
#define SSIZE 8510   /* tamaño de la pila para cada tarea */

struct tcb* tarea_p[SLAVES];   /* array de punteros a bloques de control */

void funcion0 ()
{
   int contador;
   for (contador=0; contador < 1000; contador++)
   {
       printf(''soy la funcion 0\n'');
   }
   OSKill(0); /* La tarea se suicida, dejando su lugar en la cola de listos libre */
}

void funcion1 ()
{
   int contador;
   for (contador=0; contador < 1000; contador++)
   {
       printf(''soy la funcion 1\n'');
   }
   OSKill(0); /* La tarea se suicida, dejando su lugar en la cola de listos libre */
}

int main()
{
  /*inicialización de la multitarea en modo con desalojo */
  OSMTInit(PREEMPT);

  /*inicialización de las tareas*/
  tarea_p[0] = OSSpawn("tarea0", funcion0, SSIZE1, MAX_PRI, 1);
  tarea_p[1] = OSSpawn("tarea1", funcion1, SSIZE1, MIN_PRI, 0);

  /*puesta a punto de las tareas*/
  OSReady(tarea_p[0]);
  OSReady(tarea_p[1]);

  /*inicialización del planificador*/
  OSPermit(); /* En este punto la monotarea retomará el control cuando la cola de listos
                 de la multitarea quede vacía, es decir, cuando se suiciden las dos tareas
  return 0;
```

```
}
```

En modo con desalojo el orden de escritura de la pantalla LCD es impredecible, dependerá del planificador. En este caso, al solo haber dos tareas, puede que el planificador haga desalojos bastante uniformes, pero cuando el número de tareas aumenta y los cuerpos de las tareas son diferentes entre sí, el orden no es predecible en principio. Es muy importante recordar que la multitarea con desalojo puede ocasionar problemas, pues las tareas se ejecutan prácticamente en paralelo. Es recomendable usar **semáforos** para acceder a cualquier recurso y así sincronizar las tareas y evitar problemas. En este ejemplo, sería recomendable utilizar un semáforo para acceder a la pantalla LCD. Para más información, consultar el apartado de semáforos en este mismo manual.

Para evitar que se le asigne procesador durante un tiempo determinado a una tarea se puede utilizar la función `OSSleep()`. De este modo se pueden implementar tareas periódicas que ejecutan una iteración, y se duermen por un cierto tiempo antes de ejecutar la iteración siguiente. Puede interesar controlar con precisión la periodicidad con que estas se ejecutan. Si el tiempo en realizar los cómputos de la iteración es despreciable entonces un sencillo esquema de `OSSleep(periodo)` es suficiente. Sin embargo si el tiempo de cómputo de la iteración no es despreciable entonces ese esquema no es válido porque el periodo real será la suma del tiempo en que está dormido y el tiempo de cómputo de la iteración. Para resolverlo podemos usar la función `OSGetTime()` al comenzar y al terminar el código de la iteración y restar los tiempos obtenidos. Por ejemplo, si queremos que una tarea tarde T centésimas de segundo desde que comienza a realizar un trabajo hasta que se le asigne otra vez el procesador, calcularemos lo que tarda en hacer el trabajo y se lo restaremos a T, haciendo después un `OSSleep(T-computo)` con el valor obtenido. Así, en total habrá utilizado un tiempo T.

Variables globales Si queremos usar variables globales que sean visibles en todas las hebras de nuestro programa para el eyebot entonces estas variables debemos *declararlas* al comienzo del programa y no dentro del bloque `main`. A estos efectos el bloque `main` se trata como una hebra más, si se declaran dentrode él, las demas hebras no verán esas variables.

A continuación se muestra un ejemplo:

```
#include "eyebot.h"
#include "string.h"
#include "stdio.h"

#define SLAVES 2;  /* numero de tareas */

struct tcb *tarea_p[SLAVES];   /*puntero a los bloques de control*/
```

```
struct tcb tarea_tcb[SLAVES]; /*bloques de control*/

/* declaración de variables */
/* las variables declaradas en esta parte si son visibles en las hebras */

int contador;
image img;
....

/* declaración de funciones */

.....

void main ()
{

/* estas variables tambien son globales, pero no serian visibles para otras hebras*/

int x;
char letra;
....

/*inicialización de las tareas o hebras*/
  tarea_p[0] = OSSpawn("tarea0",.......);
  tarea_p[1] = OSSpawn("tarea1",.......);
....

}
```

Sin embargo esas variables globales conviene *inicializarlas* dentro del `main`. Hemos descubierto que si la inicialización está exclusivamente al comienzo del programa, fuera del `main` o de la función de cualquier otra hebra entonces la inicialización sólo tiene efecto la primera vez que se ejecuta el programa. La segunda ejecución esa variable global se inicia con el valor que dejara la anterior ejecución. Para corregir esto conviene poner la inicialización de esas variables en el bloque `main`, aunque su declaración sea global.

3.4.2. Semáforos

Los semáforos se utilizan para coordinar las distintas tareas. El acceso a ellos para modificar su estado requiere su inicialización a través de la función OSSemInit(sem,valor). Una vez inicializados pueden ser subidos o bajados con las funciones OSSemP(sem) y OSSemV(sem) respectivamente.

Formato Función	*P.Entrada*	*P.Salida*	*Descripción*
OSSemInit (sem, valor)	Puntero al semáforo. Valor inicial	Ninguno.	Inicializa el semáforo con el valor inicial.
OSSemP (sem)	Puntero al semáforo	Ninguno.	Operación down del semáforo (Espera).
OSSemV (sem)	Puntero al semáforo	Ninguno.	Operación up del semáforo (Levanta).

Cuadro 12: Funciones para los semáforos

Formato Función	*P.Entrada*	*P.Salida*	*Descripción*
OSAttachTimer (escala, función)	Valor para la preescala del temporizador. Función que será llamada periódicamente.	Manejador para referenciar al IRQ-slot.	Adjunta una interrupción a la lista de irq ajustando su periodo de petición con la escala indicada.. Función es lo que ha de hacer.
OSDeatachTimer (Manejadortiempo)	Manejador.	0= manejador no válido. 1= OK.	Lo quita de la lista de irq.

Cuadro 13: Funciones para los temporizadores

3.4.3. Temporizadores

Es posible incluir y eliminar interrupciones en el programa gracias a las funciones OSAttachTimer (escala,función) y OSDeatachTimer (interrupción).

3.5. Acceso a los elementos de interacción

3.5.1. Pantalla

El programa que corre en el Eyebot puede acceder a la pantalla para realizar la escritura de cadenas de texto (`LCDPrintf(cadena)`), de números enteros (`LCDPutInt (entero)`), números reales (`LCDPutFloat(real)`), o el valor hexadecimal de un entero (`LCDPutHex(numero)`).

Además también es posible representar por pantalla las imágenes que se obtienen con la cámara (`LCDPutGraphic(imagen)`), y dibujar líneas (`LCDLine (x1,y1,x2,y2,color)`) y rectángulos (`LCDArea(x1,y1,x2,y2,color)`). La función `LCDPutImage((BYTE*)&imagen)` muestra la imagen de 128x64 binarizada.

Se permite el borrado total de la pantalla con la función `LCDClear()` y la visualización de 4 cadenas de 4 caracteres cada una, a modo de etiquetas de los botones situados debajo con `LCDMenu(cadena1,....,cadena4)`. .

1. int **LCDPutImage**(BYTE *)
   La función recibe como entrada una imagen de 128x64 bits. Esta imagen tiene que estar contenida en un vector de 16x64 posiciones y cada posición debe ser un byte. De este modo, cada pixel de la pantalla se corresponde con un bit del vector (16x8 = 128 pixels cada fila).

Si se quiere mostrar en la pantalla las imágenes capturadas por la cámara a pantalla completa, se podría usar esta función. Las imágenes provenientes de la captura tienen una resolución de 82x62, por tanto, habría que hacer una conversión de la imagen de 82x62 a 128x64 antes de llamar a la función `LCDPutImage`. Por ejemplo:

```
image    imagen_bn;
colimage imagen_col;

/* Declaración de la variable para la imagen de 128x64.*/
typedef BYTE image_XXL [64 * 16];
image_XXL imagen_XXL;

/* Inicializar toda la imagen a negro (zero padding) */
for(i=1; i <= 16 * 64; i++)
imagen_XXL[i] = 255;

CAMGetColFrame(&imagen_col,0);
IPColor2Grey (&imagen_col, &imagen_bn);

for(i = 0; i < imagerows; i++)
     for(j = 0; j < imagecolumns; j++)
  {
    pos = i * 128 + j;
            offset = pos % 8;
    pos = pos / 8;
            num = 128;
            for (k = 0; k < offset; k++)
                num = num >> 1;
    if (imagen[i][j] < umbral)
        imagen_XXL[pos] = imagen_XXL[pos] | num;
            else
                imagen_XXL[pos] = imagen_XXL[pos] & (255-num);
          }

LCDPutImage(imagen_XXL);
```

El bucle central crea la imagen binarizada. Para ello recorre todos los pixels de la imagen que viene de la cámara, teniendo en cuenta, que ahora cada pixel corresponde a un byte. Este byte representa un pixel en escala de grises. En la pantalla solo se pueden distinguir dos colores: blanco o negro, por tanto, analizaremos cada byte para interpretarlo como blanco o negro. Esto se realiza comparando cada byte de la imagen de la cámara con un umbral.

La imagen se captura de la cámara con `CAMGetColFrame` y se convierte a escala de grises con `IPColor2Grey`. También se podría capturar directamente en blanco y negro desde la cámara.

3.5.2. Teclado

Hay funciones que permiten la interacción del usuario con el robot a través del teclado, gracias a ellas es posible reconocer la tecla que se ha pulsado (KEYGet()), de manera que se puede bloquear el flujo de ejecución hasta que se pulse una tecla especificada (KEY-Wait(tecla)).

3.5.3. Audio

El micrófono y el altavoz incorporados en el Eyebot posibilitan la grabación y reproducción de sonidos a través de varias funciones que veremos a continuación.

1. int **AUCaptureMic**(void)
   capta el volumen en el ambiente y lo devuelve transformado en un entero de 10 bits, es decir, da al sonido ambiente un valor entre 0 y 1023. Esta función suele usarse para ver si el micrófono funciona.

2. int **AURecordSample**(BYTE* buf, long len, long freq)
   almacenará a partir de la posición apuntada por 'buf' tantos bytes como indique el entero largo 'len'. Los 28 primeros forman la cabecera, que recogerá información acerca del sonido grabado; el resto representan el sonido que la función ha grabado a través del micrófono con una frecuencia igual al entero largo 'freq'. El entero devuelto representa la frecuencia real con la que se ha grabado el ejemplo. Se trata de una función no bloqueante, es decir, ejecuta en paralelo al programa desde el cual la hemos llamado. Si no nos interesa que ocurra esto, podemos usar la función AUCheckRecord para simular un funcionamiento bloqueante. Dado que el micrófono del eyebot es muy simple, la calidad del sonido grabado es muy baja. Esto es debido a que la función principal del micrófono es la de detectar variaciones cercanas del sonido, no la de analizarlas. Si a pesar de todo queremos grabar un sonido de calidad más alta, podemos conectar al eyebot un pequeño micrófono estándar de 8 Ohmios.

3. int **AUCheckRecord**(void)
   devolverá 0 si cuando la llamamos se está grabando algo por el micrófono, y distinto de 0 en caso contrario.

4. int **AUPlaySample**(char* sample)
   reproduce el sonido 'sample' en modo no bloqueante, pero podemos simular bloqueo

Formato Función	*P.Entrada*	*P.Salida*	*Descripción*
LCDPrintf (texto)	Formato, cadena y parámetros.	Ninguno.	Imprime el texto en el LCD (versión simplificada del printf).
LCDClear	Ninguno.	Ninguno.	Limpia el LCD.
LCDPutChar (char)	Carácter a escribir	Ninguno.	Imprime el carácter en la posición del cursor.
LCDSetChar (fila, columna, char)	Carácter a escribir. Número de la fila (0-15). Número de la columna (0-6)		Escribe el char el la posición indicada.
LCDPutString (string)	Cadena a escribir.	Ninguno.	Imprime el string a partir de la posición actual del cursor.
LCDSetString (fila, columna, string)	Cadena a escribir. Número de la fila (0-15). Número de la columna (0-6).	Ninguno.	Escribe el string en la posición dada.
LCDPutHex (entero)	Entero que será escrito.	Ninguno.	Escribe el entero en formato hexadecimal.
LCDPutHex1 (entero)	Entero que será escrito.	Ninguno.	Escribe el número como un byte hexadecimal.
LCDPutInt (entero)	Entero que será escrito	Ninguno.	Escribe el número en decimal.
LCDPutIntS (entero, espacios)	Entero que será escrito. Número de espacios.	Ninguno.	Escribe el número poniendo espacios delante si es necesario.
LCDPutFloat (real)	Número que será escrito.	Ninguno.	Escribe el real.
LCDPutFloatS (real, spaces, decimales)	Número que será escrito. Mínimo número de espacios. Número de decimales después del punto	Ninguno.	Escribe el real con espacios delante y con los decimales indicados.
LCDMode (modo)	Modo de display deseado. (NON)SCROLLING. (NO)CURSOR.	Ninguno.	Modo: SCROllING (las líneas se desplazan hacia arriba), NONSCROLLING (al completar la pantalla se borra todo lo anterior), NOCURSOR (no se muestra la posición actual con el cursor), CURSOR (se muestra la posición actual con el cursor).
LCDSetPos (fila, columna)	Fila (0-6). Columna (0-15).	Ninguno.	Coloca el cursor en la posición dada.
LCDGetPos (fila, columna)	Punteros a enteros que almacenarán la fila y la columna.	Fila actual (0-6). Columna actual (0-15).	Devuelve la posición en la que está el cursor.
LCDPutGraphic (imagen)	Imagen en blanco y negro.	Ninguno.	Escribe la imagen en blanco y negro en el LCD empezando por la esquina superior izquierda. Sólo son escritos 80x54 pixels.
LCDPutColorGraphic (colorimag)	Imagen en color.	Ninguno.	Escribe la imagen en color empezando por la esquina superior izquierda. Sólo son escritos 80*54 pixels. CUIDADO: utilizando esta función se destruye el contenido de la imagen.

Cuadro 14: Funciones para la pantalla

Formato Función	*P.Entrada*	*P.Salida*	*Descripción*
LCDPutImage (img BYTE)	Imagen en blanco y negro (128*64 pixels).	Ninguno.	Imprime la imagen en toda la pantalla.
LCDMenu (string1, .., string4)	Strings para el menú sobre las teclas (4 caracteres máximo). deja la entrada como estaba. “ “ blanquea la entrada.	Ninguno.	Escribe encima de la tecla correspondiente el string.
LCDMenuI (Pos, string)	Posición (1-4). Cadena (4 caracteres máximo)	Ninguno.	Escribe el string en la posición indicada.
LCDSetPixel (fila, columna, entero)	Fila (0-63). Columna (0-127). entero: 0=limpia el pixel, 1=pone el pixel, 2=invierte el pixel.	Ninguno.	Dependiendo del entero trata el pixel especificado.
LCDInvertPixel (fila, columna)	Fila (0-63). Columna (0-127).	Ninguno.	Invierte el pixel especificado.
LCDGetPixel(Fila, columna)	Fila (0-63). Columna (0-127).	0= pixel limpio 1= pixel puesto	Devuelve el valor del pixel seleccionado.
LCDArea (x1, y1, x2, y2, color)	(x1,y1) (x2,y2) x1¡x2 y1¡y2 color: 0=blanco 1=negro 2=imagen negativa		Pinta un rectángulo del color especificado. Arriba izquierda = (0,0) abajo derecha = (127,63)
LCDLine(x1, y1, x2, y2, color)	(x1,y1) (x2,y2) x1¡x2 y1¡y2 Color: 0= blanco 1= negro 2= en negativo.	Ninguno.	Dibuja una línea de (x1,y1) a (x2,y2) usando el algoritmo de Bresenham. Esquina superior izquierda = (0,0). Esquina inferior derecha = (127,63).

Cuadro 15: Funciones para la pantalla (II)

Formato Función	*P.Entrada*	*P.Salida*	*Descripción*
KEYGetBuf (letra)	Puntero a un carácter.	Carácter con la tecla pulsada (KEY1, KEY2, KEY3, KEY4 de izq a der).	Espera que se presione una tecla y la almacena en letra.
KEYGet	Ninguno.	Código de la tecla pulsada (KEY1, KEY2, KEY3, KEY4 de izq a der)	Devuelve el valor de la tecla pulsada.
KEYRead	Ninguno.	Código de la tecla pulsada (KEY1, KEY2, KEY3, KEY4 de izq a der) 0 si no se ha pulsado ninguna.	Lee la tecla y la devuelve pero no espera.
KEYWait(codtecla)	Código de la tecla a esperar (KEY1, KEY2, KEY3, KEY4 de izq a der) o ANYKEY para cualquiera.	Ninguno.	Espera hasta que se presiona la tecla especificada.

Cuadro 16: Funciones para el teclado

empleando la función AUCheckSample. Devuelve 0 si el formato del ejemplo a reproducir no es soportado por el altavoz del eyebot, y distinto de 0 en caso contrario. En la tabla inferior podemos ver los formatos soportados. Lo más frecuente es pasar a esta función un puntero a BYTE (recordemos que en C un carácter es en realidad 1 byte). Por lo tanto, 'sample' podrá ser el sonido resultante de grabar con la función AURecordSample o cualquier otra variable declarada en nuestro programa, ya sea externa o inicializada en el mismo, y su calidad dependerá de su contenido. Es interesante destacar que podemos grabar sonidos y convertirlos a formato hexadecimal en el PC, con lo que podríamos bajar sonidos al eyebot de mayor calidad. Existe una herramienta que facilita este trabajo en el PC y que se puede conseguir en la siguiente dirección:

`http://robotics.ee.uwa.edu.au/eyebot/ftp/ROBIOS/pc-appl/audio/`

En ella encontraremos el fichero auconvert.c, que al compilarse generará el a.out. Podemos ejecutar este último fichero de la siguiente manera:

```
> ./a.out nombre\_archivo\_de\_sonido
```

Cada vez que hagamos esto, crearemos un fichero de nombre hexdump.txt que contendrá el equivalente hexadecimal de nombre_archivo_de_sonido.

5. int **AUCheckSample**(void)
   devolverá 0 si cuando la empleamos todavía se está reproduciendo algún ejemplo, y distinto de 0 en caso contrario.

6. int **AUTone**(int freq, int msec)
   reproduce un tono con una frecuencia en Hz 'freq' durante un tiempo en ms 'msec' de forma no bloqueante. Podemos simular bloqueo mediante el uso de la función AUCheckTone.

7. int **AUBeep**(void)
   es un caso particular de AUTone, pero con una frecuencia y una duración predefinidas, por lo que no requiere el paso de argumentos.

8. int **AUCheckTone**(void)
   devolverá 0 si cuando la empleamos se está reproduciendo algún tono, y distinto de 0 en caso contrario.

A continuación, mostramos una pequeña demo que usa todas estas funciones:

Formato Función	*P.Entrada*	*P.Salida*	*Descripción*
AUPlaySample (sample)	Dato a reproducir.	Sonidos indicados. 0 = si no es un tipo soportado	Reproduce un sample en modo no bloqueante. Formatos soportados: WAV o AU/SND (8bits, pwm o mulaw) a 5461,6553,8192, 10922,16384, 32768 Hz.
AUCheckSample	Ninguno.	FALSE mientras el sample está siendo reproducido.	Test no bloqueante.
AUTone (freq, msecs)	Frecuencia del tono (65-21000 Hz). Longitud del tono (msecs) (1-65535).	Tono.	Reproduce un tono con la frecuencia dada y durante el tiempo especificado.
AUCheckTone		FALSE mientras el tono está siendo reproducido	Test no bloqueante.
AUBeep	Ninguno.	Ninguno.	reproduce un BEEP.
AUCaptureMic	Ninguno.	Valor del micrófono (entero de 10 bits).	Coge un sonido del micrófono.
AURecordSample (buffer, longitud, freq)	Buffer de almacenamiento. Longitud (número de bytes + 28 bytes de cabecera). Frecuencia deseada.	Frecuencia real.	Graba un ejemplo del micrófono del sistema y lo almacena en buffer. Formatos: AU/SND (pwm) con 8 bits sin signo.
AUCheckRecord	Ninguno.	FALSE mientras se está grabando.	Test no bloqueante.

Cuadro 17: Funciones para el audio

```
#include <eyebot.h>
#include <stdlib.h>

int main ()
{
  char c1,c2;
  BYTE* buffer = (BYTE*) malloc (30000);
  extern BYTE Ready[];
  /* Voz que dice "Ready". La variable Ready será un array de bytes que está
     definido en otro archivo */
  int condicion = 1;

  LCDPrintf(" Demo de Audio ");
  LCDPrintf("-----------------------");
  LCDMenu("Tono","Mic","WAV","END");

  while(1)
  {
     c1 = KEYRead();
     switch (c1)
     {
      case KEY1:
     AUTone(1000,500);
         while(!AUCheckTone()){}
         break;
      case KEY2:
     AUBeep();
     while (condicion)
     {
            LCDClear();
            LCDMenu("Live", "Rec", "Play", "END");
            LCDPrintf(" Demo Microfono: \n");
            c2 = KEYRead();
            switch (c2)
            {
      case KEY1:
  LCDMenu("","","","STOP");
  LCDSetString(2,0, "Volumen: ");
  while (KEYRead()!= KEY4)
  {
                 LCDSetPos(2,8);
```

```
                    LCDPrintf("%d\n", AUCaptureMic());
                    OSWait(20);/* CPU bloqueada 20 centésimas */
                }
                break;
            case KEY2:
                LCDSetString(2,0, "Grabando");
                AURecordSample(buffer, 33000, 11000);
                /* Como a buffer le vamos a ir asignando memoria dinámicamente,
                   podemos hacer que el sonido grabado ocupe más bytes de los
                   indicados en la declaración de la variable buffer */
                while (!AUCheckRecord()){}
                break;
            case KEY3:
                LCDSetString(2,0, "Reproduciendo");
                AUPlaySample(buffer);
                /* Reproduciremos los 33000 bytes que ocupa buffer tras haber
                   grabado algo previamente */
                while(!AUCheckSample){}
                break;
          case KEY4:
          free(buffer);
                  condicion = 0;
          break;
              default: break;
           }/* switch (c2) */
       }/* while (condicion) */
    break;
     case KEY3:
        if(AUPlaySample(Ready))
            while(!AUCheckSample()){};
        else
        {
            LCDClear();
            LCDPrintf("Error: formato incorrecto.");
        }
    break;
     case KEY4:
    return 1;
     default: break;
     } /* switch (c1) */
} /* while (1) */
```

Formato Función	*P.Entrada*	*P.Salida*	*Descripción*
OSSetTime (horas, min, seg)	Valor para hora minutos y segundos.	Ninguno.	Fija el reloj del sistema a la hora dada.
OSGetTime (horas, min, seg, ticks)	Punteros a enteros para almacenar horas, minutos, segundos y ticks (1segundo = 100 ticks)	Horas, minutos, segundos y ticks del sistema.	Devuelve la hora del sistema.
OSShowTime	Ninguno.	Ninguno.	Muestra el reloj del sistema en el display.
OSGetCount	Ninguno.	Número de centésimas de segundo desde la última inicialización.	Es un int de 32 bits, hasta 248 días aproximadamente.
OSWait (tiempo)	Tiempo en centésimas.	Ninguno.	Espera ocupando CPU durante el tiempo indicado.

Cuadro 18: Funciones de manejo tiempo

```
} /* main */
```

3.6. Otros recursos del sistema

3.6.1. Manejo del tiempo

1. int **OSShowTime**(void)
   muestra el reloj del sistema en el display con formato: "the time is: hhmmsscc", donde hh=horas, mm=minutos, ss=segundos, cc=centésimas.

El siguiente ejemplo es un programa que muestra continuamente la hora en la pantalla del eyebot, con un intervalo de refresco de 50 centésimas:

```
#include <eyebot.h>
  int main(void)
  {
    do
    {
        LCDClear();
        LCDMenuI(4,"FIN"); /*Muestra botón de fin */
        OSShowTime();
        OSWait(50);

    }while (KEYRead() != KEY4);
    return 0;
  }
```

Formato Función	*P.Entrada*	*P.Salida*	*Descripción*
IPLaplace (imageor, imagendest)	Imagen de entrada.	Imagen de destino.	Calcula la función de Laplace al origen.
IPSobel (imagenor, imagendest)	Imagen de entrada.	Imagen de destino.	Aplica la función de Sobel.
IPDither (imagenor, imahendest)	Imagen de entrada.	Imagen de destino.	Operador Dither con un patrón 2x2.
IPDiffer (imactual, imganterior, imdest)	Imagen actual. Imagen anterior.	Imagen de destino.	Calcula la diferencia en escala de grises entre la imagen actual y la anterior pixel a pixel.
IPColor2Grey (imagcolor, imdest)	Imagen en color de entrada.	Imagen en niveles de gris de salida	Convierte una imagen en color a niveles de gris (4-bit).

Cuadro 19: Funciones para el procesamiento de imágenes

Formato Función	*P.Entrada*	*P.Salida*	*Descripción*
BYTE OSReadInLatch (int latchnr)	(latchnr) Número del buffer de entrada deseado (rango: 0..3)	Estado actual del buffer de entrada.	Lee el contenido del buffer de entrada seleccionado.
BYTE OSWriteOutLatch (int latchnr, BYTE mask, BYTE value)	(latchnr) Número del buffer de salida deseado (rango: 0..3) (mask) Máscara de bits para realizar un and lógico. (value) Máscara de bits para realizar un or lógico	Estado previo del buffer de salida	Modifica un buffer de salida y mantiene el estado global. Ejemplo: OSWriteOutLatch (0, 0xF7, 0x08); fija bit4 Ejemplo: OSWriteOutLatch (0, 0xF7, 0x00); limpia bit4
BYTE OSReadOutLatch(int latchnr)	(latchnr) Número del buffer de salida deseado (rango: 0..3)	Estado actual del buffer de salida.	Lee el contenido actual del buffer de salida

Cuadro 20: Funciones para el acceso a los buffers de I/O de bajo nivel

3.6.2. Procesado de imagen

A las imágenes obtenidas con la cámara es posible aplicarles distintos filtros como la función de Laplace (IPLaplace (origen,destino)), un filtro Sobel (IPSobel (origen,destino)) o un operador Dither (IPDither (origen,destino)). También es posible el cálculo de la diferencia en escala de grises entre dos imágenes IPDiffer(actual,anterior), y la conversión de una imagen en color a escala de grises (IPColor2Grey (origen,destino)).

3.6.3. Acceso a los buffers de I/O de bajo nivel

El acceso a los buffers de entrada y salida de bajo nivel se puede hacer a través de las funciones habilitadas para su lectura (OSReadOutLatch (latch) y OSReadInLatch (latch) para el de salida y el de entrada respectivamente). La escritura en el buffer de salida se hace con la función OSWriteOutLatch (latch,máscara,valor).

Formato Función	*P.Entrada*	*P.Salida*	*Descripción*
int OSGetAD (int channel)	(channel) Canal AD deseado. Rango: 0..15	Valor del muestreo 10 bits.	Captura un valor simple de 10 bits del canal AD especificado. El valor es almacenado en los bits menos significativos de los 32 bits del entero.
int OSOffAD (int mode)	(mode) 0= Desactivación completa 1 = Desactivación rápida.	Ninguno.	Desactiva los 2 Conversores AD (ahorrando energía).Una llamada a OSGetAD vuelve a activar los Coversores AD.

Cuadro 21: Funciones de acceso al conversor A/D

3.6.4. Conversor analógico digital

El acceso al conversor A/D permite tanto capturar un valor del mismo(OSGetAD(canal)) como activar y desactivar el conversor(OSOffAD(modo)).

3.6.5. Información del sistema

A través de las funciones implementadas para facilitar el acceso al sistema se puede realizar consultas acerca de valores que conciernen al mismo, como por ejemplo, el número de versión del sistema operativo, su hora, la velocidad del microprocesador, el tipo de robot sobre el que se está ejecutando el RoBIOS (con la función OSMachineType()).

También es posible acceder a modificar alguno de esos datos, como por ejemplo la hora (OSSetTime (horas,minutos,segundos)), o detener la ejecución del programa que se esté ejecutando gracias a la función OSWait (tiempo).

4. Utilización del PC para programar el eyebot

Antes de construir programas para el eyebot hay que preparar el entorno adecuado en el ordenador personal. Esta preparación incluye tres pasos: la instalación del compilador cruzado, la copia del sistema operativo del eyebot y la preparación del puerto serie. Las librerías y cabeceras que proporciona el sistema operativo del eyebot deben copiarse en el PC para que los programas que escribamos para el eyebot se puedan compilar y enlazar convenientemente.

Cualquier programa que escribamos para el eyebot lo editaremos en el PC, con cualquier editor de texto (por ejemplo vi, emacs o xemacs). Utilizamos *lenguaje ansi-C*, por su versatilidad y porque disponemos de un compilador cruzado para la plataforma motorola-68k del eyebot. Una vez escrito el código fuente de nuestro programa lo compilaremos en el PC para que ejecute en el eyebot utilizando el compilador cruzado. Automáticamente se enlaza con las librerías que ofrece el sistema operativo del eyebot y se genera el ejecutable *.hex*. Una vez obtenido el ejecutable se descarga en el eyebot a través del puerto serie.

Formato Función	*P.Entrada*	*P.Salida*	*Descripción*
Misc			
OSVersion	Ninguno.	Versión del SO.	Devuelve el valor de la RoBIOS que esta corriendo.
OSError (mensaje, numero, bool dead)	Mensaje. Número. Dead: 0= espera a pulsar una tecla (no deadend) 1= detiene el procesador (deadend).	Ninguno.	Imprime el mensaje y el número en el LCD y para el proceso o lo detiene hasta que se pulse una tecla.
OSMachineType	Ninguno.	Tipo del hardware utilizado (VEHICLE, PLATFORM, WALKER).	Devuelve el tipo de máquina.
OSMachineSpeed	Ninguno.	Velocidad actual (Hz).	Informa sobre la velocidad del microprocesador.
OSMachineName	Ninguno.	Nombre del Eyebot.	Dice el nombre del Eyebot (introducido en HDT).
OSMachineID	Ninguno.	Identificador del Eyebot.	Informa del identificador del Eyebot (HDT).
Interrupciones			
OSEnable	Ninguno.	Ninguno.	Activa todas las interrupciones de la CPU.
OSDisable	Ninguno.	Ninguno.	Desactiva todas las interrupciones de la CPU.
Variables guardadas para tpuram			
OSGetVar (posición)	Posición para salvar el valor. Valores: SAVEVAR1-4 para palabras SAVEVAR1a/1b-4b para bytes	Valor salvado. 0- 65535 para palabras. 0-255 para bytes	Dada una posición de tpuram, devuelve el valor salvado en la posición dada.
OSPutVar (posición, Valor)	Posición donde salvar el valor. Valor: SAVEVAR1-4 para palabras. SAVEVAR1a/1b-4b para bytes. Valor a salvar: 0- 65535 para palabras. 0-255 para bytes.	Ninguno.	Guarda en la posición dada el valor. Normalmente SAVEVAR1-3 es ocupado por RoBIOS.

Cuadro 22: Funciones de información del sistema

Seguiremos un convenio de numeración de versiones para facilitar el desarrollo iterativo de los programas y la coordinación entre varios programadores.

4.1. Instalación en el PC del compilador cruzado

El compilador cruzado para el eyebot se llama **gcc68**, para instalarlo en el PC conviene seguir los siguientes pasos. En las descargas de ficheros desde internet la *dirección base* donde está todo lo necesario es www.ee.uwa.edu.au/~braunl/eyebot/ftp.

1. Crear un directorio `directorioeyebot` donde va a estar ubicado el compilador. En este directorio también copiaremos posteriormente las librerías del sistema operativo. Por ejemplo el `directorioeyebot` puede ser `/usr/local/eyebot` si se va a instalar en una máquina para que lo utilicen todos sus usuarios, o puede ser un directorio cualquiera de tu cuenta, por ejemplo el usuario federico crea el directorio `/users/federico/eyebot` si lo quiere instalar exclusivamente para él.

2. Bajar los ficheros gcc68linux.tgz, g++68linux.tgz y libc-include.tgz de la *dirección base*/GCC/ y copiarlos en `directorioeyebot`. Descomprir los ficheros anteriores en `directorioeyebot`, con lo que se crean los subdirectorios gcc-m68k, g++-m68k e include respectivamente.

3. Descargar todos los ficheros de *dirección base*/ROBIOS/cmd/linux y copiarlos en `directorioeyebot/bin`.

4. Una vez hecho esto hay adaptar algunos ficheros a nuestra configuración particular. Dentro de `directorioeyebot/bin` debe revisarse y modificarse el fichero gcc68 . También habilitar permisos de ejecución (con el comando chmod) a gcc68, dl y srec2bin. En concreto hay que verificar que las líneas respectivas contienen los siguientes valores:

   export CFLAGS= "$WARN -I. -I$mc/include -I$libc $CPU -O -msoft-float $opts"
   export basedir= `directorioeyebot/`
   export gccdir = $basedir/g++-m68k
   export gccparts = $gccdir/lib/gcc-lib/m68k-coff/2.95.2 (según la versión que tengamos)
   $basedir/bin/srec2bin

   Además de estas modificaciones, en el script gcc68 conviene añadir la ruta completa en las llamadas a los comandos `basename` y `rm`, de manera que quedan /usr/bin/basename y /bin/rm respectivamente.

5. El último paso consiste en incorporar a la variable de entorno PATH de cada usuario el directorio `directorioeyebot/bin`. Para conseguir esto hay que editar el fichero .bashrc (y el .bash_profile) que está en el directorio HOME de cada usuario y añadir la línea PATH=$PATH:/`directorioeyebot/bin`. Ojo, para que los cambios tengan efecto hay que reiniciar la sesión.

4.2. Copiar sistema operativo del eyebot en el PC

El sistema operativo del eyebot se llama **RoBIOS**, la última versión incorporada es la 4.2 (de 2001-07-16). Para copiarlo conviene seguir los siguientes pasos:

1. Bajarse de la *dirección base* la última versión de RoBios (por ejemplo robios42usr.tgz) y dejarla en `directorioeyebot`.

2. Descomprimir en ese mismo directorio con lo que aparece el subdirectorio mc, que contiene todo lo relativo al sistema operativo RoBIOS.

Si el script `directorioeyebot/bin/gcc68` está bien configurado el compilador buscará las cabeceras de RoBIOS en `directorioeyebot/mc/include` y el enlazador buscará las librerías de RoBIOS en `directorioeyebot/lib/include`. Si esto no funcionara bien entonces hay que forzar los enganches, habrá que copiar recursivamente el directorio `directorioeyebot/mc/include` en `directorioeyebot/include` y el directorio `mc/lib/include` en `directorioeyebot/libc`.

4.3. Descarga por el puerto serie

El programa **dl** permite descargar en el eyebot el fichero con el programa ejecutable.Si se quiere descargar el fichero `file.hex` hay que ejecutar `dl file.hex`. Para que funcione correctamente el usuario que lanza dl debe tener permisos de escritura en el dispositivo linux del puerto serie (por ejemplo /dev/ttyS0). dl está en `directorioeyebot/bin` y debe tener permisos de ejecución concedidos.

Puede ocurrir que haya problemas a la hora de descargar archivos al robot, que no se logre establecer la comunicación. Si esto sucede se puede solucionar comentando en los ficheros gass68, gcc68 y gld68o que hay en `/directorioeyebot/bin` las siguientes líneas:

```
echo srec2Binary
srec2bin $name.hex
```

Estas líneas tienen como función comprimir el fichero .hex resultante tras la compilación, y descargar al eyebot la versión comprimida. Su uso no es imprescindible porque el envio sin comprimir es válido, por lo tanto pueden ser omitidas. OJO comentarlas sólo cuando provocan la descarga incorrecta. Si funcionan bien es mejor tenerlas descomentadas porque la descarga del programa es mucho más rápida.

4.4. Probando la instalación con el programa holamundo

Para verificar que la preparacion del entorno en el PC funciona correctamente proporcionamos un programa llamado holamundo.c:

```
#include <eyebot.h>

#define MENSAJE "Hola Mundo\n"

int main()
{
 LCDPutString(MENSAJE);
 return 0;
}
```

Este programa es muy sencillo y se limita a imprimir en la pantalla del eyebot el mensaje "Hola mundo". Prueba a compilarlo en el PC y descargarlo en el eyebot. Para compilarlo invoca el comando `gcc68 holamundo.c`, debe generar un fichero llamado holamundo.hex, que es el ejecutable correspondiente. Para descargarlo en el eyebot conéctalo al puerto serie, ponlo a recibir el fichero (se explica en la sección 2.3) y en el PC ejecuta el comando `dl holamundo.hex`.

4.5. Creando un programa para el robot

4.5.1. Escribir el código fuente

El lenguaje de programación utilizado es C complementado con funciones especiales para el eyebot. A la hora de programar es muy útil tener a mano un listado con todas estas funciones. Estas funciones son las que se describen a lo largo de todo este manual y proporcionan el acceso por programa a los distintos recursos del robot.

Normalmente el flujo de control de un programa sobre el robot móvil estará organizado a partir de dichas funciones. De este modo las distintas tareas que debe realizar se materializan como un sistema multitarea con desalojo.

Esta descomposición de la aplicación es el primer paso. Dos tareas muy comunes son el chequeo de las comunicaciones y el refresco de la interfaz gráfica. A estas dos se le unen las tareas específicas de cada programa.

Otros esquemas del flujo de control también son posibles, por ejemplo un bucle infinito de percepción, procesamiento y acción de control.

Si el programa accede a distintos recursos hardware como los sensores, los motores, la cámara, etc., o a recursos hardware como el control VW, ha de asegurarse que primeramente se han inicializado estos recursos.

Como ya hemos visto la plataforma software empleada ofrece funciones que ya vienen implentadas para inicializar, acceder y liberar tanto los recursos hardware como software.

Adicionalmente es posible para el programador encontrar nuevas soluciones para otro tipo de control en el movimiento, para el control de la cámara, etc. Se trata pues de un sistema abierto, con muchas posibilidades de expansión.

4.5.2. Compilar y enlazar en el PC

4.5.3. Descargar en el eyebot el ejecutable

Una vez que se ha escrito es programa es necesario guardarlo con la extensión '.c' y compilarlo con el compilador cruzado que se habrá instalado previamente (ver sección anterior). Para ello sólo es necesario escribir 'gcc68 nombre_programa.c '. Esto generará un fichero 'nombre_programa.hex' , que será el que debamos descargar al Eyebot. Existen otras posibilidades a la hora de escribir los programas, podremos usar lenguaje ensamblador (compilaremos con el script gas68), o bien una combinación de C y ensamblador, compilando cada uno con el script que corresponda y utilizando gld68o para construir el ejecutable (hay que tener cuidado ya que algunos criterios de gnu para lenguaje ensamblador difieren del estandar de Motorola). Para descargar este programa hay que actuar tanto en el lado del PC como en el del Eyebot.

Pasos para el Eyebot En el menú principal del Eyebot deberemos pulsar "Hrd" , después pulsaremos "Set" y en la siguiente pantalla "Ser" (serial). Nos aparecerá la pantalla para hacer el setup del puerto serie. Para modificar valores nos fijaremos en el valor marcado con un asterisco (*) , pulsando + y - modificaremos dicho valor. La opción Interface indica el interface por el que se va a realizar la transmisión, utilizando el cable serie deberemos seleccionar 1.

En la opción Speed deberemos elegir la velocidad adecuada. Ésta deberá coincidir con la velocidad que pusimos en nuestro fichero dl, en nuestro caso 115200. Con la opción Handshake se habilita (o deshabilita) que se indique por pantalla del número de bytes que se han transmitido. Nosotros dejaremos RTS/CTS.

Para validar estas selecciones pulsaremos "END" 3 veces (hasta llegar a la pantalla principal).

Para descargar el programa pulsaremos "Usr" en la pantalla principal, entraremos en una pantalla llamada User Program. Dentro de ella seleccionaremos "Ld" (Load). El Eyebot se quedará esperando el programa. En este paso es cuando en el PC se ha de ejecutar la instrucción dl mi_programa.hex . Hecho esto comenzará la descarga. Durante la transmisión el programa de descarga muestra el nombre del programa que está siendo cargado al Eyebot y un indicativo del número de bytes recibidos. El resultado final es una pantalla que indica que la descarga se ha efectuado correctamente y ofrece la oportunidad de ejecutar el programa (pulsando "Run"). Si durante la descarga se produce algún error en la LCD aparecerá un mensaje indicando el error que se ha producido. Para cargar el programa el error deberá ser subsanado y deberemos ejecutar otra descarga. Un listado

con los mensajes de error, su causa y su posible solución se puede encontrar en la URL http://www.ee.uwa.edu.au/~braunl/eyebot/dl/ exceptions.html

Pasos para el PC En el lado del PC entre otras posibilidades utilizaremos el script dl. Este programa permite descargar un fichero al eyebot a través del puerto serie. El aspecto de este fichero es el siguiente:

```
#!/bin/csh -ef
# Download application via COM1 to EyeBot

stty -F /dev/ttyS0 speed 115200 raw >/dev/null
echo Transmitting at 115200
cat $1 > /dev/ttyS0
```

En este fichero cada usuario deberá seleccionar sus preferencias y adaptarse a su equipo. Nostros tenemos el COM1 en el dispositivo /dev/ttyS0, y hemos decidido descargar a una velocidad de 115200 baudios. Es posible descargar programas también a través del COM2 y a distintas velocidades (9600,19200,38400,57600,115200 son los valores posibles). También hay que comprobar (1ª linea del script) que dl está configurado para nuestra shell y que tenemos habilitado el permiso de ejecución.

4.6. Creando una librería para el robot

Para generar una librería que pueda ser usada por muchos programas del eyebot primero hay que generar el código objeto de la librería. No hay enlazado dinámico, es una librería estática, de hecho sólo es código objeto que se enlaza con el objeto del programa principal que usa la librería.

1. Crear los ficheros .h y .c que corresponden al código fuente (cabeceras e implementación) de la librería.

2. Generar el/los fichero objeto `.o` a partir de los anteriores. Para conseguir esto es necesario utilizar el script `gcc68o` en lugar de `gcc68`. La sintaxis es la siguiente: `gcc68o libreria.c`

3. Una vez generados todos los ficheros objeto, incluído el del fichero que contiene al programa principal, debemos enlazarlos para crear el ejecutable `.hex`. El ejecutable es el único fichero que realmente se descarga en el robot. Para enlazar se utiliza el script `gld68o`. Por ejemplo:
   `gld68o fichero_que_contiene_main.o fichero1.o fichero2.o fichero3.o ...`

4. El último paso será cambiar el nombre al ejecutable (si nos apetece), puesto que el enlazador habrá creado `fichero_que_contiene_main.o:.hex`. Para ello bastará con

hacer:
`mv fichero_que_contiene_main.o:.hex fichero_que_contiene_main.hex`

4.7. Numeración y empaquetamiento de versiones

Cuando se genera un gran número de programas y de mejoras continuas de estos programas se hace indispensable un esquema de numeración que permita orientarse en un mar de versiones y versiones de los distintos programas. El esquema de numeración empleado consta de un nombre y dos numeros, el mayor y el menor, separados por un punto. Por ejemplo `eem-4.0-tgz` es el paquete con la apliación eem, versión 4.0. El primer número identifica cambios relevantes en una apliación respecto de la anterior versión, mientras que el segundo refleja nuevas versiones con mejoras pequeñas, no esenciales. La experiencia nos ha mostrado que es muy ventajoso adoptar un esquema de este tipo, en el cual las sucesivas mejoras se incorporan a un programa en forma de nuevas versiones.

Todo lo necesario para que determinado programa pueda compilarse y ejecutarse debe empaquetarse en una versión. En el paquete no se deben incluir las librerías comunes, porque ya estarán instaladas en todos los ordenadores. Estos paquetes se generan y se descomprimen con la herramienta `tar` de linux-unix. Facilitan la programación de mejoras sobre una versión ya existente y la portabilidad de las aplicaciones. Por ejemplo para editar desde cualquier ordenador basta traerse por la red la versión comprimida desde el repositorio, desempaquetarla en un ordenador cualquiera y compilarla o editarla en ese nuevo ordenador. Por ello es conveniente empaquetar junto con los programas básicos aquellos otros que hayan servido para comprobar alguna tarea, por ejemplo un programa que lea todos los datos que se le envíen y los muestre por pantalla, así como cualquier otro fichero que pueda ser útil en un futuro para otras personas que no hayan participado del desarrollo.

Otro consejo útil es tener siempre un backup del trabajo hecho. Uno se da cuenta de esto cuando una lamentable casualidad destruye el trabajo de muchos meses y no hay manera de recuperarlo. Un buen truco es copiar las versiones empaquetadas en dos ordenadores distintos o grabar periódicamente CDs con nuestros programas.

5. Simulador del eyebot

El simulador del eyebot se materializa tiene la forma de una librería. Esta librería, que se llama `libeyebot.a`, implementa las funciones que ofrece ROBIOS en el robot real. De esta forma el diseñador puede generar sus programas para el robot y enlazarlos con esta librería para probarlos en un escenario simulado antes de ejecutarlos en el robot real.

Además de la librería el simulador incluye un script sencillo de compilación `ee` que se encarga de compilar nuestro programa y enlazarlo con la librería de simulación, no con la librería `libeyebot.a` preparada para el robot real, que tiene el mismo nombre.

No todas las funciones de ROBIOS están soportadas en el simulador, aunque sí la mayoría (en la página oficial del eyebot[2] se especifica cuales están soportadas en la versión actual del simulador). Por ejemplo el acceso crudo a los encoders o a los motores no está implementado. En su lugar se ha simulado el conjunto la librería VW. El soporte a la cámara es opcional y se decide en tiempo de instalación si se materializan las funciones relativas a la cámara o no. Si se quiere soportar es necesario incluir unas librerías adicionales (Mesa,Imlib y Coin) para enlazar con ellas.

5.1. Instalación

En este apartado describimos el proceso de instalación para la versión eyesim-3.2, que descargamos desde la página oficial del eyebot.

Primero intentamos la instalación estándard, que viene descrita dentro del paquete eyesim-3.2. Las instrucciones establecen tres pasos con sendos comandos de shell: (1) configure (2) make (3) make install. Es decir, configuración del paquete, compilación e instalación final.

El primer paso no trajo problemas. Simplemente hay que cambiar en el fichero `configure` la etiqueta `--prefix=PATH`,donde PATH es el camino al directorio donde deseas instalar el simulador. `configure` es un script de shell que averigua la configuración del sistema concreto donde vamos a instalar el simulador y genera un `makefile` adecuado a la configuración en cada directorio del paquete. Puedes dar valores iniciales a las variables de `configure` retocando CC=c89 CFLAGS=-02 LIBS=-lposix en el propio script.

En un principio queríamos instalar sólo la parte de los motores e infrarrojos dejando de lado la cámara, que se traducen en la librería libeyebot.a. Sin embargo con los pasos (1) y (2) no compilaba bien, quizá porque los `makefile` generados automáticamente en la configuración no funcionan. Así que decidimos compilar *a mano* la librería, haciendo nuestro propio `makefile`, creando los ficheros objeto y uniéndolos todos para crear `libeyebot.a`. Después de algunos problemas decidimos incluir también el soporte para la cámara.

Para compilar la librería lo primero es conseguir las librerías de las que depende a su vez. El simulador depende de:

- Xforms. Librería para la creación de gráficos en X-Window como botones, cursores, canvas, diales. Está como paquete Debian precompilada, su código fuente no está liberado. La version que instalamos fue la 0.89. Se utiliza por ejemplo para emular la consola del eyebot.
- Mesa. Librería clónica de OpenGL, para visualización de gráficos 3D. Se puede encontrar en `http://mesa3d.sourceforge.net/`) aunque también está como paquete Debian.
- Imlib (`http://www.rasterman.com/imlib.html`)

[2] `http://www.ee.uwa.edu.au/~braunl/eyebot/`

- Coin (`http://www.coin3d.org/`)

Una vez instaladas esas librerías compilamos y enlazamos el código fuente de nuestra librería, generando la librería `libeyebot.a` que contiene al simulador. Finalmente hay que retocar el programa `ee` para que encuentre y enlace el código fuente del usuario con esa librería.

El programa `ee` llama al compilador `gcc` y enlaza con las librerías del eyesim, creando un ejecutable `output.sim`, que corre completamente en el PC con el simulador. OJO, pueden surgir problemas en tiempo de ejecución a la hora de encontrar las librerías dinámicas, por lo que tendrás que configurar al sistema para que encuentre dinámicamente las librerias Mesa, ImLib, Coin, etc que necesite. Esto se hace modificando como root el fichero `/etc/ld.so.conf` para incluir una línea con el directorio donde estén esas librerías dinámicas y ejecutando el comando `ldconf`.

5.2. Fichero de configuración

Cuando se genera un ejecutable `output.sim` para el simulador debe acompañarse en ese mismo directorio de un fichero de configuración `eyesim.p`. En este fichero se puede configurar:

- el tamaño del robot dentro del mundo simulado, así como sus velocidades máximas.
- características de sus sensores infrarrojos (PSD).
- qué mundo virtual cargar en el simulador (vienen dados como ficheros vrml o maze)
- donde situar, dentro del mundo simulado, al eyebot inicialmente.
- uno o varios robots en el mundo (todos corren el mismo programa).

Referencias

[O'Sullivan97] J. O'Sullivan, K. Haigh, G. Armstrong, *Xavier manual*, Robot Learning Lab, Computer Science Department, Carnegie Mellon University, Abril 1997.

[Rwi96a] Real World Interface, Inc.*B21 Users guide*, `http://www.rwii.com`.

[Rwi96b] Real World Interface, Inc.*System software and RAI-1 documentation*, `http://www.rwii.com`.

[García-Alegre93] M.C. García-Alegre, A. Ribeiro, J. Gasós y J. Salido, *Optimization of fuzzy behavior_based robots navigation in partially known industrial environments*, IEEE International Conference on Industrial Fuzzy Control & Intelligent Systems, Houston TX, pp 50-54, 1993. ISBN: 0-7803-1485-9.

[Braunl] Eyebot[3] Prof. Thomas Bräunl, The Univ. of Western Australia. `http://www.ee.uwa.edu.au/~braunl/eyebot/`.

[García01] Esther García, *Construcción de un teleoperador para el robot eyebot*, Proyecto Fin de Carrera 2002, Universidad Carlos III, Madrid.

[Cañas00] José María Cañas, *Manual del robot móvil Hermes*, Informe Técnico, Instituto de Automática Industrial, CSIC.

[3] `http://www.ee.uwa.edu.au/~braunl/eyebot/`