UNIVERSIDAD CARLOS III DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

Dpto.de INFORMÁTICA

INGENIERÍA TÉCNICA INDUSTRIAL
ELECTRÓNICA INDUSTRIAL
PROYECTO FIN DE CARRERA

CONSTRUCCIÓN DE UN TELEOPERADOR PARA EL ROBOT EYEBOT

Tutor: Camino Fernández Llamas

Director: Jose María Cañas Plaza

Autor: Esther García Morata

Enero 2.002

Dedicado a todos los que confiaron en mí.

Agradecimientos

Este proyecto no hubiera sido posible sin la colaboración de todo el grupo de robótica de la Universidad Rey Juan Carlos, especialmente Jose María Cañas y Vicente Matellán, a quienes les agradezco la confianza que depositaron en mí desde el primer momento, ofreciéndome la posibilidad de trabajar con ellos.

Gracias a Camino Fernández, ella fue quien me puso en contacto con el grupo, y a Javier Coso e Ignacio Gutiérrez por su paciencia, su apoyo y su tiempo.

A mis padres les debo el ofrecerme las oportunidades que ellos no tuvieron y a mis amigos el ánimo para seguir adelante.

Muchas gracias a todos.

Resumen

Eyebot es un robot móvil sobre el que el grupo de robótica de la Universidad Rey Juan Carlos está desarrollando sus investigaciones. El robot dispone de una serie de sensores y actuadores que le permiten desenvolverse en el entorno. Cuenta con un microprocesador Motorola 68332 y entre otras cosas dispone de tres sensores infrarrojos, una cámara digital y dos encoders, uno para cada motor. Además tiene dos motores que permiten el movimiento del robot y dos servos, uno para mover la cámara y el otro para el pateador. El sistema operativo del robot incorpora una consola a través de la cual se pueden realizar testeos de los distintos elementos hardware y se pueden descargar programas al robot. Este sistema operativo ofrece también una interfaz de programación a través de la cual se puede acceder a los distintos dispositivos desde los programas del usuario.

Este proyecto aborda la implementación de un sistema teleoperado para el robot Eyebot. El sistema ha sido concebido como una arquitectura cliente-servidor y consta de dos programas, uno que se ejecuta en el robot (el programa servidor) y otro que lo hace en el PC (el programa cliente). El cliente se conecta al servidor para solicitarle información de los distintos sensores y para enviarle comandos de movimiento para sus actuadores. Por su parte el programa servidor responde a las peticiones de información y ejecuta los movimientos que se le ordenan. Para construir el sistema ha sido necesario programar un protocolo de comunicaciones que comunica el cliente y el servidor. El cliente incorpora una interfaz gráfica con la capacidad de representar al robot, su entorno y su movimiento, incluyendo las opciones necesarias para que el usuario solicite la transmisión de los datos recogidos por los sensores del robot. Esta interfaz incorpora además las herramientas necesarias para que el usuario pueda gobernar los movimientos del robot.

Todo esto ha sido programado bajo una plataforma Linux, utilizando las herramientas tanto de creación de entornos gráficos como de programación que se facilitan con este sistema operativo para la implementación de los requisitos solicitados.

Índice General

Índice de Figuras

Índice de Tablas

Capítulo 1

Introducción

Desde los años 70 hasta hoy la robótica ha dejado de ser una fantasía literaria para convertirse en una realidad. Los primeros avances fueron dificultosos y tímidos, pero su desarrollo ha experimentado un crecimiento progresivo en las ultimas décadas. Dentro de la robótica un tema que despierta gran interés es el de los robots móviles. Este tipo de robots se caracterizan por tener patas, ruedas, orugas o cualquier otro elemento que les permita desplazarse por el entorno. Esta característica hace que la realización de proyectos orientados a controlar la navegación espacial autónoma se haya elevado considerablemente. Algunos de estos proyectos tienen características que los hacen atractivos y permiten desarrollarse en un marco propicio la competición. Por ejemplo, las competiciones de sumo o la RoboCup, que es una competición de fútbol para robots.

Durante el año 2000 el grupo de robótica de la Universidad Rey Juan Carlos planteó la creación de un equipo de fútbol para participar en la RoboCup dentro de la categoría de tamaño medio. Lo primero que se pensó es en el tipo de robots que se iban a utilizar, los cuales debían de tener unas dimensiones reducidas para adaptarse a la normativa de la competición.

Las posibilidades principales eran dos, la compra de los mismos o la fabricación propia. Esta última alternativa fue descartada ya que el grupo no disponía de los medios necesarios. Además se deseaba tener un equipo en un espacio de tiempo relativamente corto. Por ello se decidió comprar los robots integrantes del equipo. A la hora de elegir el modelo entre los muchos disponibles en el mercado las posibilidades se reducían a cuatro: el robot Tritt, los LEGO Mindstorms, el robot Khepera y el soccerbot de Eyebot. Las diferencias de precio entre los cuatro son considerables y cada uno de ellos cuenta con distintas características. Principalmente se buscaba un modelo que ofreciera una solución completa, es decir, que estuviera listo para ser utilizado de inmediato, su

precio no fuera excesivo y que a ser posible tuviera una cámara.

Robot	*Cámara*	*Solución completa*	*Precio*
Tritt	No	No	Bajo
LEGO Mindstorms	No	No	Bajo
Khepera	Adaptable	Sí	Muy elevado
Soccerbor (Eyebot)	Sí	Sí	Medio

Tabla 1.1: Características de los distintos robots

Se decidió comprar robots Eyebot [13] [5] por ser el único modelo que incluía la cámara incorporada, con las ventajas de que era una solución que venía lista para funcionar y tenía un precio razonable.

Este robot dispone de ciertos motores que le permiten moverse por el espacio de forma autónoma y como está pensado para jugar al fútbol lleva incorporado un pateador para poder dar al balón. Incorpora tres sensores de infrarrojos para detectar obstáculos cercanos, dos encoders para medir el desplazamiento del robot y una cámara que le permite captar imágenes del entorno.

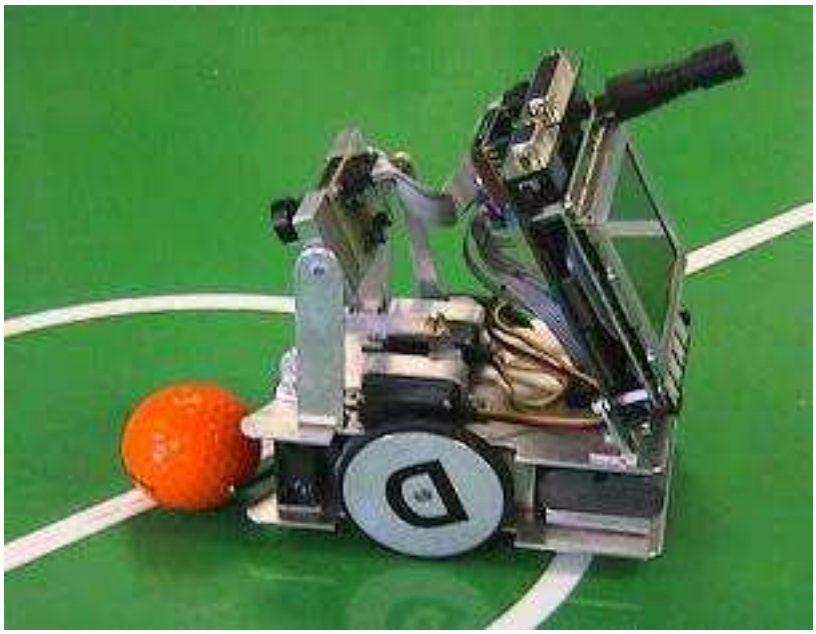


Figura 1.1: Eyebot (imagen suministrada por el fabricante)

Una vez que se disponía de los robots era necesario aprender a manejarlos para sacarles el máximo rendimiento. En esta línea surgió el proyecto de la creación de

un teleoperador para el robot como primer paso para familiarizarse con los sensores y actuadores del robot, así como con el acceso a ellos desde programas.

Por teleoperación se entiende el control de un robot situado en una localización remota [1] [9]. Este concepto general engloba los aspectos de la teleactuación y la telesensorización. La teleactuación es la faceta de la teleoperación relacionada con la generación remota de órdenes a los actuadores del robot. La telesensorización es el aspecto relacionado con la captación y visualización de la información sensorial del robot. En un sistema teleoperado la persona encargada de controlar el robot ha de saber en todo momento el valor que recogen los sensores del robot, visualizando sus valores de algún modo [6]. Ambas partes están separadas físicamente y pueden no tener un contacto visual directo, por tanto únicamente en función de los datos sensoriales representados el operario toma la decisión de generar los distintos comandos de movimiento que ejecutará el robot.

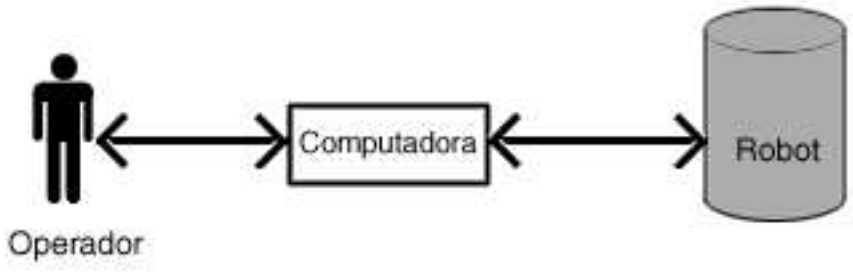


Figura 1.2: Diagrama de un sistema teleoperado

La teleoperación de robots surge por la necesidad humana de acceder a entornos peligrosos o de difícil acceso para el hombre, como el interior de volcanes, zonas de alta radiación en centrales nucleares, inmersiones marítimas en aguas a baja temperatura o alta profundidad, etc. Actualmente hay proyectos comerciales que se han desarrollado específicamente para acceder a este tipo de entornos, como el robot móvil teleoperado desarrollado por Iberdrola [7] dentro del proyecto SRT, que se encarga de acceder a zonas de muy alta radiación en centrales nucleares y de realizar determinados trabajos de mantenimiento dentro de ellas. Este tipo de robots incorpora sistemas que permiten

la recuperación del robot en caso de que se produzca algún fallo, como por ejemplo en la alimentación. El entorno donde se va a desenvolver el Eyebot objeto de este proyecto no es peligroso ni de difícil acceso, pero la creación de un teleoperador para el mismo es una buena herramienta para conocer a fondo la arquitectura hardware y software del robot, para así poder determinar sus cualidades y establecer sus limitaciones.

El robots Xavier [8] de la Universidad Carnegie Mellon y el robot Hermes del Instituto de Automática Industrial [10] [3] son plataformas de investigación para navegación autónoma. A estas plataformas se les ha implementado un teleoperador con el objetivo de estudiar el funcionamiento de sus elementos hardware y desarrollar otros proyectos paralelos. Estos teleoperadores incorporan unas interfaces gráficas a través de las cuales se visualizan los valores de los distintos sensores del robot. En ellas la representación de los sensores de hace de manera egocéntrica, con el robot como centro de referencia. En cuanto a la teleactuacion, la interfaz del robot Xavier incorpora unos botones para mover el robot de manera básica en traslaciones y giros. Por el contrario para el robot Hermes en la interfaz se incluye un *joystick* que permite realizar un control más amplio en los movimientos permitiendo traslaciones y giros combinados. En las imágenes 1.3 y 1.4 se muestran estos dos ejemplos de interfaces gráficas.

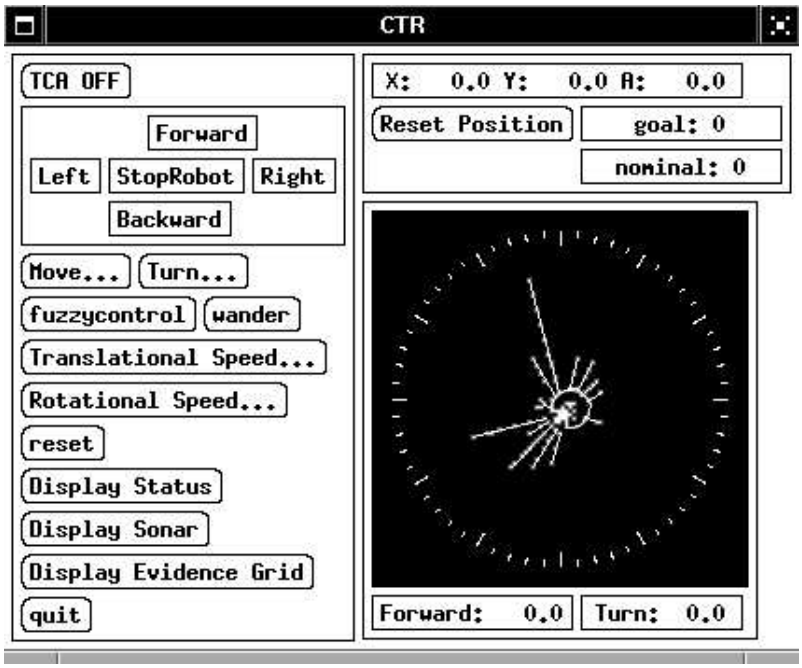


Figura 1.3: Interfaz gráfica para el robot Xavier. Ventana egocéntrica con sensores sónar y botones de movimiento

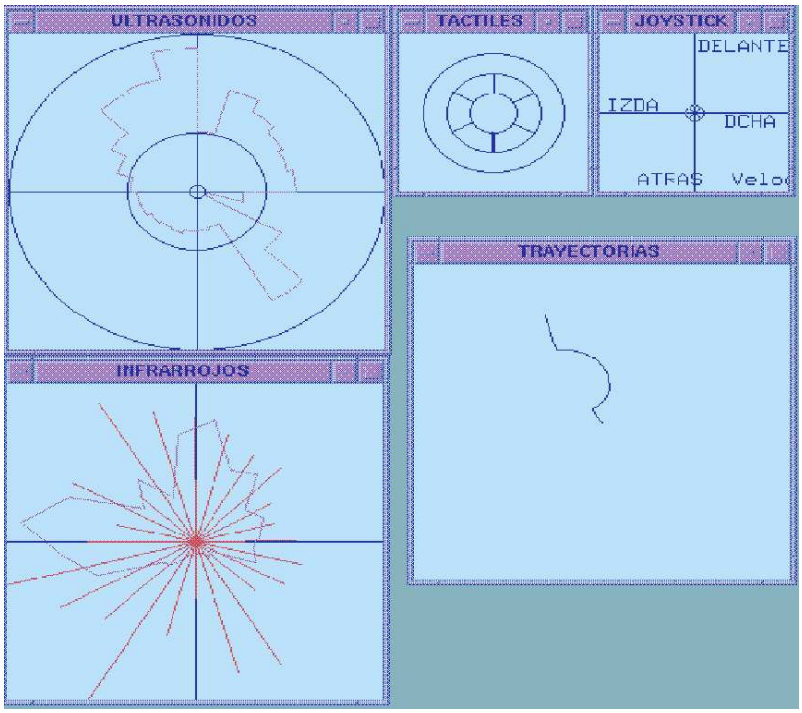


Figura 1.4: Interfaz gráfica para el robot Hermes. Ventana egocéntrica con ultrasonidos y joystick

1.1 Objetivo

El objetivo principal de este proyecto es la construcción de un teleoperador para el robot Eyebot, desde el cual de manera remota se puedan visualizar los valores devueltos por todos los sensores del robot a través de una interfaz gráfica que ofrezca elementos para poder generar ordenes de movimiento para el robot [4].

La implementación del teleoperador requiere de un conocimiento del robot que se va a teleoperar. Este conocimiento incluye tanto la arquitectura hardware y software del Eyebot como el entorno de programación del mismo. El fabricante proporciona adicionalmente un sistema operativo que incluye una interfaz de programación para acceder a los distintos sensores y actuadores del robot desde el programa que en él se ejecuta.

El teleoperador ha de incluir una interfaz gráfica que permita al usuario humano la visualización de la información sensorial del robot y le ofrezca una herramienta para interaccionar con los actuadores del robot. Esta interfaz ha de ser intuitiva, capaz de ser utilizada por un usuario no experto.

Para que sea posible la teleoperación tanto el programa que se ejecuta en el robot

como el que se ejecuta en el PC han de comunicarse. Es necesaria la creación de un sistema de comunicaciones y determinar un protocolo de manera que ambas partes puedan entenderse. Este sistema de comunicaciones debe de ser capaz de enviar y recibir la información sensorial y los comandos de movimiento.

1.2 Arquitectura propuesta

Se propone un sistema basado en una arquitectura cliente-servidor similar a la utilizada en la teleoperación del robot Hermes [10]. El sistema está dividido en dos procesos que se comunican entre sí. Para el robot Hermes la comunicación se realizaba utilizando un protocolo TCP/IP, para el teleoperador del Eyebot será a través del interfaz RS-232. En el sistema teleoperado de Hermes el programa cliente y el servidor se ejecutaban en sendos PCs. En este caso las arquitecturas donde se ejecutan estos programas son diferentes. Por un lado hay un servidor ejecutándose en el Eyebot que se encarga detectar el establecimiento o el cierre de una conexión, de enviar la información sensorial requerida y ejecutar los movimientos que se le indican.

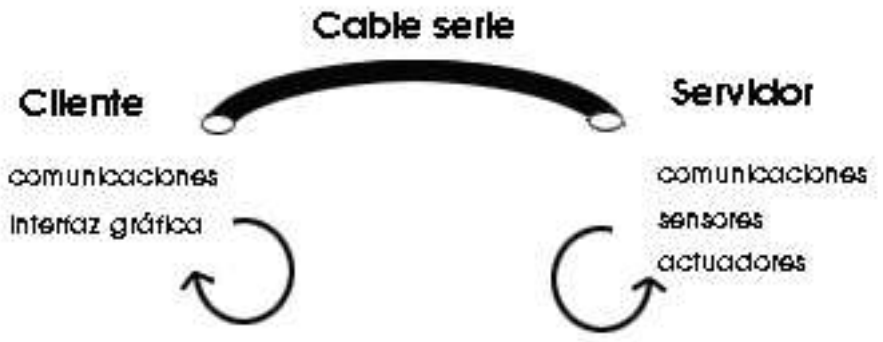


Figura 1.5: Arquitectura cliente-servidor

Por el otro lado hay un cliente ejecutándose en el PC. El cliente ha de establecer la comunicación con el servidor y visualizar los valores de los distintos sensores. Además el cliente ha de enviar las órdenes de movimiento al robot. Estas órdenes serán indicadas por un usuario humano a través de una interfaz gráfica incorporada en el cliente.

La conexión entre el cliente y el servidor se realiza por el puerto serie. El robot posee también la capacidad de comunicarse vía radio, pero este enlace limita las velocidades de transmisión. Estas velocidades son importantes cuando se va a transmitir un gran

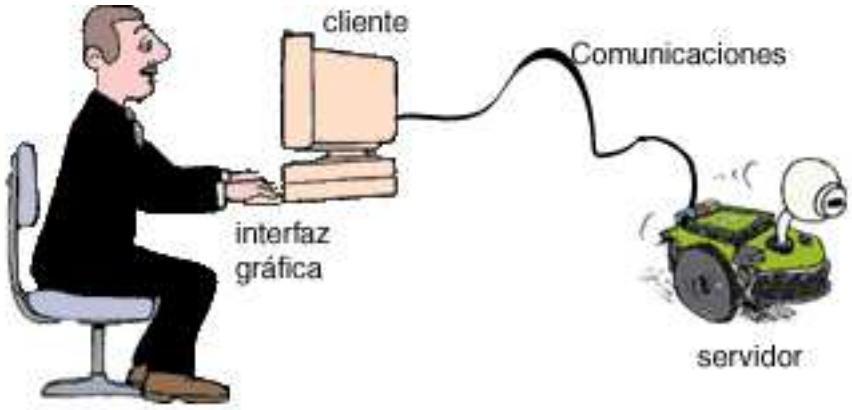


Figura 1.6: Arquitectura propuesta

volumen de datos, como es el caso del envío de imágenes desde el servidor al cliente, por ello se ha optado por preferir velocidades de transmisión más elevadas y acotar la libertad de movimientos (el uso del puerto serie requiere que el cliente y el servidor estén conectados a través de un cable, mientras que la comunicación vía radio no requiere de éste).

Tanto el cliente como el servidor han de ejecutar distintas tareas de modo independiente. Esto obliga a la utilización de distintas herramientas de multiprogramación. Para el programa cliente en el Eyebot la API del sistema operativo ofrece dos modos distintos de crear programas multihebra: de manera cooperativa y el modo con desalojo. En el PC se utilizará la librería Rai-Scheduler que ha sido la utilizada con anterioridad por el grupo.

Para generar los programas a descargar en el robot se utilizará un compilador cruzado gcc68 sobre una plataforma Linux. El lenguaje usado para programar al robot es ANSI-C, pudiendo ser también utilizado el lenguaje ensamblador del M68000. C también es el lenguaje elegido por el fabricante del Eyebot para codificar sus librerías de la interfaz de programación.

El sistema operativo sobre el cual se ejecutan el entorno de desarrollo y el programa cliente es Linux, un sistema similar a Unix, de libre distribución, multitarea y multiusuario, de 32 bits muy robusto y ágil. 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.

1.3 Estructura de la memoria

La presente memoria se articula en seis capítulos en los que se describe el robot tanto a nivel hardware como software y se aborda la implementación de un sistema teleoperado con arquitectura cliente-servidor basada en los programas que se ejecutan en el Eyebot (el servidor) y en el PC (el cliente).

Las herramientas utilizadas para la implementación del programa servidor que se ejecuta en el Eyebot son descritas en el capítulo 2, donde además se hace una descripción de todos los componentes hardware del robot, se estudia el sistema operativo del mismo y la interfaz de programación que ofrece.

En el capítulo 3 se aborda la implementación del programa servidor, explicando tanto el código, como el protocolo de comunicaciones empleado, necesario para que el cliente y el servidor puedan entenderse.

Las herramientas utilizadas a la hora de realizar la programación del cliente se estudian en el capítulo 4, incluyendo una descripción del uso del puerto serie, empleado para las comunicaciones entre el PC y el robot. En este capítulo se hace una descripción de la librería Rai-Scheduler, elegida para implementar la multiprogramación. La interfaz gráfica que ofrecerá el programa cliente al usuario estará construida con la librería XForms, por lo que también se hace una descripción de sus elementos gráficos y de su uso general.

En el capítulo 5 se describe la implementación del programa cliente, tanto en su interfaz gráfica, como en su código interno, con explicaciones de la representación del sistema real sobre la interfaz.

Las conclusiones de esta memoria y los trabajos futuros a los que el presente proyecto da lugar corresponden al sexto y último capítulo.

Capítulo 2

Eyebot

En este capítulo se hace una descripción del robot que se va a teleoperar desde el PC, tanto en sus elementos físicos, sensores y actuadores, como en el software que permite acceder a ellos y manipularlos desde programa.

El robot consta de una placa con un microprocesador a la que se han conectado distintos sensores y actuadores. Los sensores son los encargados de proporcionar la distinta información del entorno en el que se desenvuelve el robot y de determinar la posición del robot en dicho entorno. El Eyebot está provisto de tres sensores de infrarrojos, dos sensores cuenta vueltas (en adelante encoders) y una cámara.

Los actuadores son los elementos que se encargan de que el robot realice alguna acción, por ejemplo desplazarse, ya sea en traslación o en giro, para lo cual se utilizan dos motores, uno para cada rueda. También es posible controlar la orientación de la cámara y del pateador a través de dos servo-motores conectados a ellos. El Eyebot cuenta además con elementos que posibilitan la comunicación con un PC o con otro Eyebot y la interacción con el usuario.

Cada vez que se enciende un Eyebot en él se ejecuta el sistema operativo del robot (en adelante RoBIOS, Robot Basic I/O System). En este sistema operativo se pueden realizar distintos test de los elementos hardware conectados al robot.

El sistema operativo del Eyebot ofrece una interfaz de programación (API) que puede ser utilizada por los diseñadores para acceder a los distintos elementos del sistema de una manera sencilla. Esta interfaz se ofrece en forma de librería con la que enlazar el código fuente del programa usuario. Estos programas se crean en el PC y desde él son descargados al robot.

En este capítulo describiremos el proceso completo para la generación de un programa para el robot, incluyendo una descripción de las acciones necesarias para la

preparación del PC de manera que pueda ser utilizado para crear programas para el robot.

Al final del capítulo y de modo ilustrativa se aborda la explicación del procedimiento general para la generación de un programa para el Eyebot.

2.1 Hardware

El robot está constituido por un microprocesador, varios sensores y varios actuadores. El microprocesador es el encargado de ejecutar los distintos programas para el robot. Los sensores recogen información del entorno, mientras que los actuadores permiten al robot desenvolverse en dicho entorno.

Dentro del robot hay tres clases distintas de sensores: los encoders, los infrarrojos (en adelante PSD, Position Sensitive Detector) y la cámara. La función de los sensores infrarrojos es detectar la presencia de un obstáculo cercano. Se han situado tres sensores de este tipo en el robot, uno a cada lado del mismo y otro en el frente. Con las variaciones en los valores de los encoders se determina si se ha producido algún cambio en la posición u orientación del robot, para ello están situados uno en cada motor. La cámara del Eyebot permite capturar imágenes en color y en blanco y negro. Está situada en la parte frontal de robot.

Los actuadores del robot permiten el desplazamiento del mismo por el entorno. Dos motores permiten al robot realizar cambios en su posición y orientación. La cámara puede ser orientada a través de un servomotor destinado para tal fin. Igualmente se puede mover el pateador para empujar la pelota.

La descarga de los distintos programas desde el PC al Eyebot se realiza a través del puerto serie. Este puerto también puede ser utilizado para la comunicación entre robots, o de un robot con un ordenador. El inconveniente de este puerto es la necesidad de un cable que conecte físicamente el emisor y el receptor. Este inconveniente está superado con la comunicación a través del módulo de radio, con el inconveniente de que las velocidades de transmisión alcanzables son menores. El puerto paralelo del robot es utilizado para el depurador externo.

Gracias a una pantalla (también LCD, Liquid Cristal Display) es posible visualizar lo que está ocurriendo en el Eyebot. En la parte inferior hay cuatro 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.

Figura 2.1: Eyebot: Vista frontal

Todos los recursos hardware del robot se pueden agrupar en varias categorías que se analizan a continuación.

2.1.1 Placa base

Consta 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. El microprocesador permite controlar dos puertos serie y un puerto paralelo. Además tiene 8 entradas digitales, 8 salidas digitales y 8 entradas analógicas, ya que el microprocesador tiene incorporando un conversor AD.

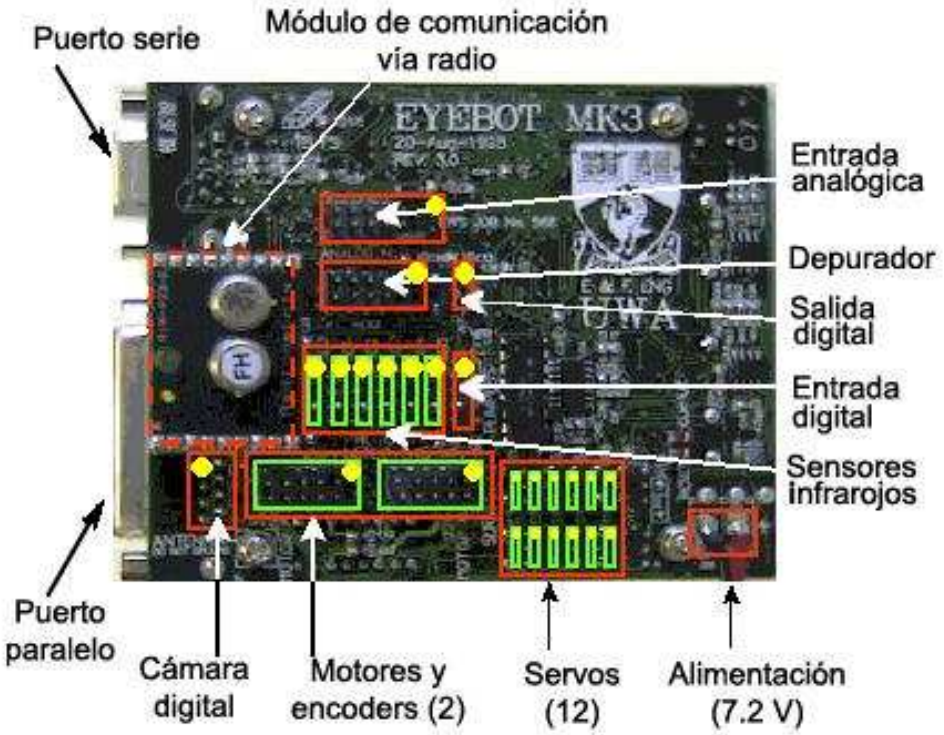


Figura 2.2: Eyebot: Vista posterior

Sus características permiten que se le puedan conectar 2 motores con sus correspondientes encoders, 12 servos, una cámara y 6 sensores de infrarrojos (PSD). Esta placa constituye el corazón del sistema y a ella se pueden conectar distintos recursos, todos ellos gobernados por el microprocesador.

2.1.2 Sensores

Los sensores que recogen información del entorno para su posterior tratamiento son los infrarrojos y la cámara. Adicionalmente el robot tiene unos encoders que informan sobre la posición y orientación del robot en dicho entorno, contando el desplazamiento que ha realizado cada rueda.

- Infrarrojos

  Los sensores de infrarrojos (del tipo GP2D02 de Sharp) miden la distancia a los obstáculos cercanos. 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. Hemos determinado empíricamente sus errores, resultando los siguientes:

Error angular: 5-7º aprox.

Error radial Eyebot: 20% - 50%

Error Frontal: 19%.

Error Izquierdo: 50%.

Error Derecho: 42%.

Nuestro robot viene equipado con tres infrarrojos, que 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.

- Encoders o cuenta vueltas
  Dos encoders situados uno en cada motor devuelven un número de pulsos. Este valor indica el desplazamiento que cada rueda ha realizado.

- Cámara
  La cámara incorporada en el Eyebot captura imágenes del entorno del robot, tanto en color como en blanco y negro. Trabaja con 24 bits en color o en escala de grises, proporcionando una resolución de 80x60 píxels. Esta resolución es suficiente para la mayoría de las tareas que realiza el robot y permite un procesamiento rápido de la imagen. Esta cámara puede ser usada sobre el Eyebot directamente o mediante un adaptador conectarla al PC, utilizando el software adecuado (Improv).

2.1.3 Actuadores

- Motores de continua

  Dos motores de continua posibilitan la movilidad del robot en un modo tipo tanque. Esto significa que para realizar los cambios de dirección es preciso que la rueda correspondiente se desplace a mayor velocidad que la otra.

  La velocidad de los motores se fija en porcentajes, con un rango de valores comprendido entre -100 a 100, negativo hacia atrás y positivo hacia delante. 0 detiene el motor.

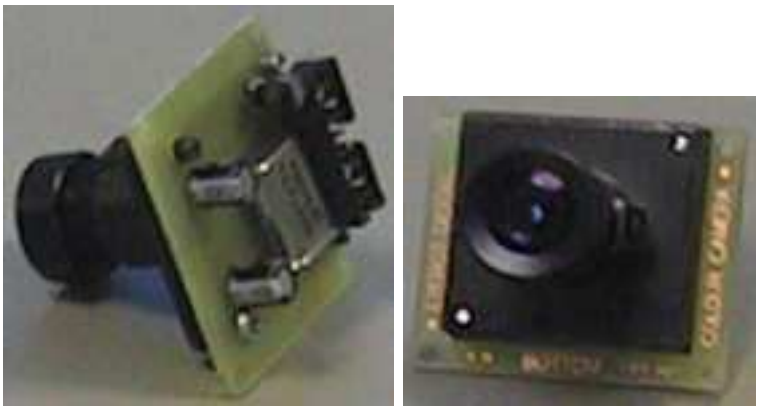


Figura 2.3: Vista posterior y frontal de la cámara

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

Los motores llevan asociados un encoder para cada uno, lo que permite poder realizar una realimentación de manera que se puede establecer algún tipo de control en lazo cerrado, como por ejemplo un control en velocidad.

- Servos

  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. Para estos servos no se incluyen sensores de posición, luego no hay posibilidad de hacer una realimentación, por lo que su posición real respecto a la indicada está diferenciada por un grado de incertidumbre.

  El motivo por el cual no se producen controles en lazo cerrado de estos motores es su uso principal son controles on/off, dónde lo único que se tiene en cuenta son dos posiciones.

  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.

  Nuestro Eyebot dispone de dos servos uno para mover la cámara y otro para el pateador.

2.1.4 Comunicaciones

El robot dispone de tres posibilidades para establecer una comunicación con el exterior, ya sea con otro robot, o con un PC. Dos de ellas se han de establecer a través de un cable que una ambas partes. Se trata de la comunicación vía puerto serie, y a través del puerto paralelo. La tercera opción es una comunicación sin cables, utilizando el módulo de radio.

- Puerto Serie
  A través de un conector RS-232 situado en la parte frontal del robot es posible conectar 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 situado 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 sin cables, ya sea entre robots, de modo que no se necesita un ordenador central, o entre el robot y un PC.

  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 con el usuario

El robot dispone de varios elementos que posibilitan la interacción del usuario con el sistema.

- Pantalla

Una pantalla sirve para la comunicación del sistema operativo con el usuario. También puede ser utilizada como dispositivo de salida para los programas que se ejecutan en el Eyebot, ya que en ella pueden escribir sus mensajes.

Físicamente es una pantalla pequeña que consta de un grid de 128 x 64 píxels, desde el que se pueden visualizar hasta 17 caracteres ASCII por línea, con un total de 8 líneas (la línea inferior está reservada para las etiquetas de menú).

Figura 2.4: Display y botonera

- Botonera
  En la parte inmediatamente inferior a la LCD hay una hilera de 4 botones. Permite al usuario elegir entre las distintas opciones que le presenta el sistema operativo o como dispositivo de entrada para los distintos programas que se ejecutan en el robot.

- Altavoces
  El Eyebot puede emitir sonidos a través de un altavoz incorporado en su placa base. Se trata de un altavoz del tipo piezo-eléctrico.

Adicionalmente es posible incorporar un altavoz externo utilizando un conector preparado para ese fin, que se encuentra situado en la parte frontal del robot. El volumen de ambos altavoces 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.

2.2 Sistema operativo

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 una consola, una tabla con los dispositivos hardware conectados y una interfaz de programación.

El sistema operativo gestiona los diferentes recursos del sistema, ofreciendo a los usuarios una consola desde la que cargar y ejecutar sus programas.

Dentro del sistema operativo hay una tabla en la que se definen los diferentes dispositivos hardware que se pueden conectar al robot, se trata de la HDT. 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.

En este apartados se hace una descripción práctica del uso de la consola para realizar test al Eyebot, preparar la descarga de programas, así como ejecutarlos. Seguidamente se explicará la función de 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.

2.2.1 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

Figura 2.5: Ejemplos de robots con la misma RoBIOS y distinta HDT

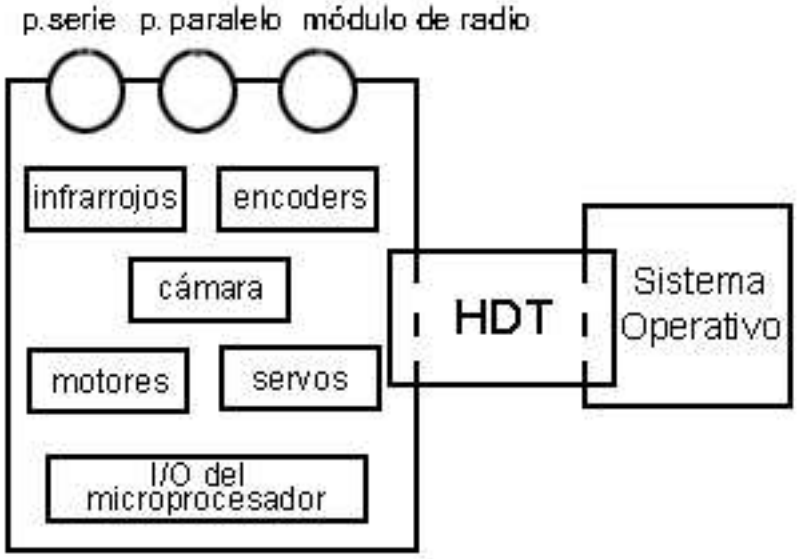


Figura 2.6: HDT

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.

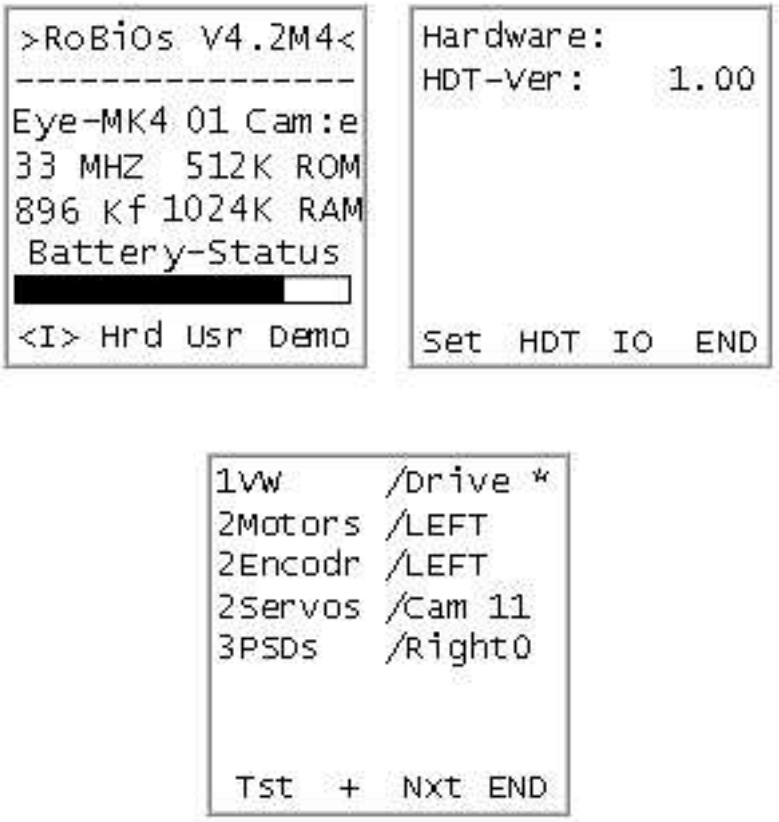


Figura 2.7: Pasos para llegar al test de los dispositivos

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 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”.

Figura 2.8: Test del control Vw

- 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”.

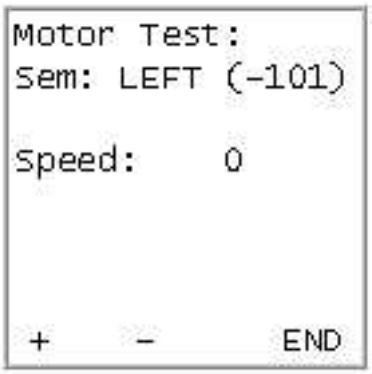


Figura 2.9: Test del motor izquierdo

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

  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

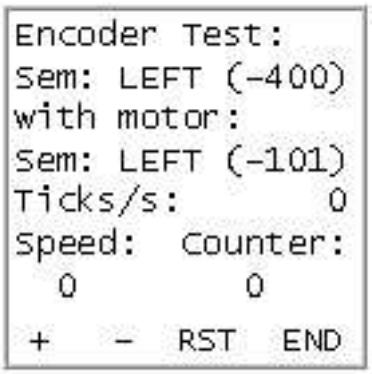


Figura 2.10: Test del encoder izquierdo

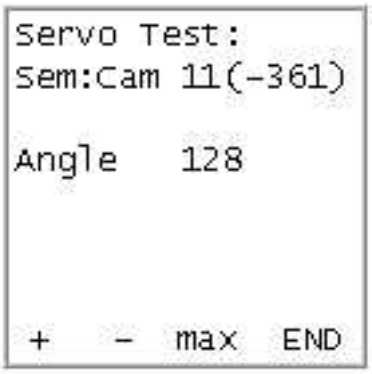


Figura 2.11: Test del servo asociado a la cámara

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”.

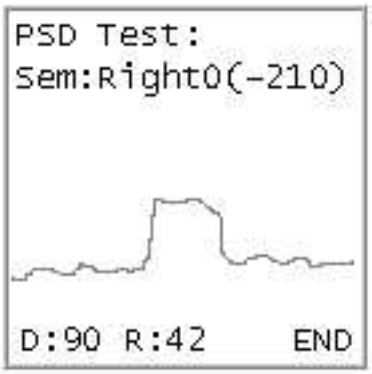


Figura 2.12: Test para el infrarrojo derecho

Carga y ejecución de programas. Además de un test 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 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.

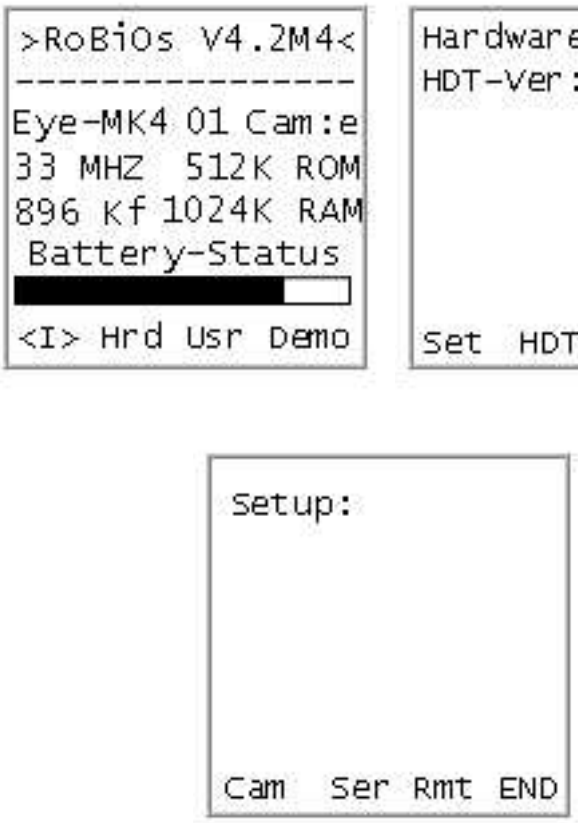


Figura 2.13: Pasos para llegar al menú de configuración del puerto serie

Figura 2.14: Configuración de los parámetros para la recepción

- 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

Figura 2.15: Modo recepción de programas

Figura 2.16: Descarga de un programa

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

Figura 2.17: Programa descargado correctamente

Figura 2.18: Manejo de los programas almacenados en la ROM

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.

2.2.2 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 hay que tenerla en cuenta al modificar la HDT. Para realizar los cambios oportunos se debe 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 es necesario 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.

2.3 Interfaz de programación

2.3.1 Acceso a los sensores

El RoBIOS ofrece diferentes funciones que posibilitan el acceso a los infrarrojos, los encoders y la cámara.

Para todos los sensores el esquema de acceso es siempre el mismo. Se ha de llamar a las funciones de inicialización para posteriormente poder trabajar con el sensor y poder resetear su estado, hacer una lectura, etc., cuantas veces se quiera. Cuando ya no se desea utilizarlo hay que liberar el recurso correspondiente.

- Infrarrojos

  Acceder a los sensores de infrarrojos 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)). El valor que se obtiene con esta función está sin calibrar, mientras que el obtenido con PSDGet está calibrado con una tabla de la HDT. La necesidad de utilizar esta tabla para obtener la distancia real a un objeto se debe a que se está trabajando con sensores cuya respuesta está muy condicionada al entorno en el que se hayan. Para modificar la tabla de calibración hay que modificar el fichero fuente de la HDT que se esté utilizando y volver a cargársela al Eyebot. Es conveniente la liberación del recurso al finalizar el programa, PSDRelease() libera todos los infrarrojos que se estén utilizando.

- 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).

  La interfaz básica de uso es ésta, pero cuando se está utilizando la librería VW (que es lo habitual), el procedimiento de acceso se ve modificado, ya que esta librería inicializa internamente los encoders. La captura de encoders mientras se utiliza el control VW se explica en detalle en la sección 2.4.6, en este mismo capítulo.

Formato Función	*P.Entrada*	*P.Salida*	*Descripción*
Sensores PSD			
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.

Tabla 2.1: Funciones para los PSD

Hay que tener en cuenta que el valor que devuelven los encoder son pulsos, y no distancia desplazada. 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).

- 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).

Formato Función	*P.Entrada*	*P.Salida*	*Descripción*
Encoders			
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{3}$2-1) 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.

Tabla 2.2: Funciones para los encoders

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 valores 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().

Formato Función	*P.Entrada*	*P.Salida*	*Descripción*
Cámara			
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 cámara.
CAMMode (modo)	Modo= (N0)AUTOBRIGHTNESS	Ninguno.	Pone la cámara en el modo seleccionado.

Tabla 2.3: Funciones para la cámara

2.3.2 Acceso a los actuadores

- 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 de la librería VW, con la que se obtiene un control realimentado en velocidad de los motores.

  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. Con la utilización de esta interfaz directa para el manejo de los motores los movimientos resultantes no serán precisos, ya que no se tienen en cuenta factores como la inercia del motor, el rozamiento de las ruedas con el suelo, etc.

  Existe otro modo de mover los motores que utiliza un control en lazo cerrado. Se trata de la librería VW, que proporciona un manejo de los motores en coor-

dinación con los encoders, de manera que el sistema realimentado obtiene unos movimientos mucho más precisos que los obtenidos con la interfaz directa. El uso de esta librería se verá en detalle en la sección 2.4.6 de este mismo capítulo.

Formato Función	*P.Entrada*	*P.Salida*	*Descripción*
Motores			
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

Tabla 2.4: Funciones para los motores

- 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.

  Los servos están orientados para su uso en control de posición. No van asociados a ninguna librería que proporcione un control realimentado, por que no hay encoders que indiquen la posición actual del servo en cada momento, lo que la precisión de su respuesta está hasta cierto punto limitada. Esto provoca que normalmente estos actuadores sean utilizados para sistemas con respuesta on/off.

2.3.3 Sistemas de comunicaciones

Los sistemas de comunicaciones permiten que el robot entre en contacto con otros robots o con un PC, de manera que se posibilite un intercambio de datos entre ellos.

Formato Función	*P.Entrada*	*P.Salida*	*Descripción*
Servos			
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

Tabla 2.5: Funciones para los servos

Estos datos deben tener una estructura definida por el protocolo de comunicaciones adoptado de manera que ambas partes, emisor y receptor sean capaces de entender los mensajes que les lleguen y emitir mensajes entendibles por la otra parte. Este protocolo de comunicaciones se estudia en detalle en el capítulo 3.

Dentro del Eyebot hay tres canales para las comunicaciones, el puerto serie, el puerto paralelo y el módulo de radio. Los dos primeros necesitan que el emisor y el receptor estén unidos a través de un cable, mientras que las comunicaciones vía radio no requieren de él.

- Puerto serie

  El programa que se ejecuta 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, la 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. La acción de lectura del puerto serie es una función no bloqueante, es decir, si no hay nada que leer el flujo de ejecución no se detiene hasta que haya algún dato, por el contrario, la acción de escritura en el puerto es una

función bloqueante, ya que una vez llamada esperará hasta que se le pase el dato que hay que enviar.

Es muy importante destacar que tanto la lectura como el envío de datos solamente puede realizarse de carácter en carácter. Dado que la velocidad de transmisión por el puerto serie no es muy elevada hay que tener cuidado con la velocidad a la que se escribe en el puerto serie, ya que si se escriben muchos datos y de manera muy rápida se puede saturar el buffer de salida. De igual manera el envío de datos al robot deberá de hacerse de manera que el robot pueda recibirlos manteniendo un determinado ritmo que permita que no se acumulen cada vez más datos, ya que se acabaría por saturar el buffer de entrada.

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 (interfaz) y OSFlushOutRS232 (interfaz) respectivamente.

- 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 escritura en él.

- Radiocomunicaciones
  El uso del puerto radio 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).

  En la finalización del programa es necesario finalizar la radiocomunicación con RADIOTerm().

2.3.4 Recursos de multiprogramación

El sistema operativo ofrece la posibilidad de que el usuario pueda hacer desarrollar la programación paralela. En este tipo de programación hay distintas hebras, y el microprocesador debe encargarse de realizar la distribución del flujo de ejecución. Para

Formato Función	*P.Entrada*	*P.Salida*	*Descripción*
Puerto serie			
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, interfaz)	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, interfaz)	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, interfaz)	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 (interfaz)	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 (interfaz)	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 (interfaz)	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 (interfaz)	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.

Tabla 2.6: Funciones para acceder al puerto serie

ello el sistema operativo contempla distintas maneras de realizar el manejo de tareas y posibilita la coordinación entre ellas por medio de semáforos.

- Manejo de tareas

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

Formato Función	*P.Entrada*	*P.Salida*	*Descripción*
Puerto paralelo			
BYTE OSReadParData (void)	Ninguna	Estado actual del los 8bits de datos del puerto paralelo	Lee el contendido del puerto 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))

Tabla 2.7: Funciones para el puerto paralelo

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.

El funcionamiento de la multiprogramación en el Eyebot ha de comenzar por una inicialización del entorno de la misma indicando el método deseado, la función OSMTInit(modo) realiza esta acción. Tras ello se deberán de inicializar las tareas con OSSpawn(nombre,comienzo,tamaño_pila,prioridad,identificador). Esta función devuelve la tarea inicializada e insertada en el planificador, pero no puesta a punto, para ello se utiliza la función OSReady(tarea). Una vez que se tengan todas las tareas inicializadas y puestas a punto hay que inicializar el planificador. OSPermit() realiza esta acción para una multiprogramación con desalojo. Hecho esto el flujo del programa se distribuye entre las distintas tareas, regresando a la función que lanzó el planificador sólo después de que se hayan eliminado todas las tareas. El siguiente ejemplo muestra los pasos comentados:

*#define SLAVES 2 /*número de tareas*/*

*#define SSIZE 8510 /*tamaño de la pila*/*

Formato Función	*P.Entrada*	*P.Salida*	*Descripción*
Radiocomunicación			
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 del 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.

Tabla 2.8: Funciones para la radiocomunicación

```
struct tcb *tarea_p[SLAVES]; /*puntero a los bloques de control*/
struct tcb tarea_tcb[SLAVES]; /*bloques de control*/
funcion0{
    while(1) printf("soy la función 0\n");
}
funcion1{
    while(1) printf("soy la función 1\n");
}
void main(){
```

Formato Función	*P.Entrada*	*P.Salida*	*Descripción*
Tareas			
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 (código)	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.
			Thread es un puntero a la estructura tbl o 0 para el thread actual

Tabla 2.9: Funciones para las tareas en la multiprogramación

```
OSMTInit(PREEMPT); /*inicialización del modo de multitarea */
/*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]);
```

*OSPermit(); /*inicialización del planificador*/*

}

- Semáforos

  Los semáforos se utilizan para coordinar las distintas tareas y garantizar el acceso en exclusión mutua a variables compartidas. 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. Si el semáforo está subido la tarea que controle no se podrá ejecutar hasta que sea bajado.

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

Tabla 2.10: Funciones para los semáforos

- Temporizadores

  Es posible incluir y eliminar interrupciones en el programa gracias a las funciones OSAttachTimer (escala,función) y OSDeatachTimer (interrupción). AL incluir una interrupción se conseguirá que un programa se ejecute periódicamente cada cierto tiempo, configurable con el parámetro escala de la función OSAttachTimer.

Formato Función	*P.Entrada*	*P.Salida*	*Descripción*
Temporizadores			
OSAttachTimer (escala, función)	Valor para la pre-escala 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.

Tabla 2.11: Funciones para los temporizadores

2.3.5 Elementos de interacción

Los elementos de interacción permiten que el usuario interactúe con el sistema de distintas maneras. Por ejemplo, la pantalla se utiliza como un elemento de salida,

donde el sistema operativo y los programas de usuario pueden acceder para escribir sus mensajes. Igualmente la emisión de sonidos es utilizada como elemento de salida, a modo de aviso en determinadas circunstancias.

La botonera puede ser utilizada por el usuario para navegar por las opciones del sistema, o como elemento de entrada tanto del sistema operativo como de los programas que sobre él se ejecutan.

- Pantalla

  El programa que se ejecuta en el Eyebot puede escribir en la pantalla cadenas de texto (LCDPrintf (cadena)), números enteros (LCDPutInt (entero)), números reales (LCDPutFloat (real)), o el valor hexadecimal de un entero (LCDPutHex (número)).

  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)).

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

- 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 (KEYWait(tecla)). Un caso típico del uso del teclado es la configuración de una tecla como tecla de finalización del programa. El programa comprobará periódicamente si se ha pulsado esa tecla, y cuando ésto ocurra finalizará.

- Audio

  El micrófono y el altavoz incorporados en el Eyebot posibilitan tanto grabar como reproducir sonidos a través de las funciones AUCaptureMic() y AUPlaySample(sonido) respectivamente. También es posible el uso de AUBeep() para reproducir un pitido a modo de aviso.

Formato Función	*P.Entrada*	*P.Salida*	*Descripción*
Pantalla			
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 píxels.
LCDPutColorGraphic (colorimag)	Imagen en color.	Ninguno.	Escribe la imagen en color empezando por la esquina superior izquierda. Sólo son escritos 80*54 píxels. CUIDADO: utilizando esta función se destruye el contenido de la imagen.

Tabla 2.12: Funciones para la pantalla

Formato Función	*P.Entrada*	*P.Salida*	*Descripción*
LCDPutImage (img BYTE)	Imagen en blanco y negro (128*64 píxels).	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 píxel, 1=pone el píxel, 2=invierte el píxel.	Ninguno.	Dependiendo del entero trata el píxel especificado.
LCDInvertPixel (fila, columna)	Fila (0-63). Columna (0-127).	Ninguno.	Invierte el píxel especificado.
LCDGetPixel(Fila, columna)	Fila (0-63). Columna (0-127).	0= píxel limpio 1= píxel puesto	Devuelve el valor del píxel 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).

Tabla 2.13: Funciones para la pantalla (II)

Formato Función	*P.Entrada*	*P.Salida*	*Descripción*
Teclado			
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.

Tabla 2.14: Funciones para el teclado

2.3.6 Librerías incluidas en RoBIOS

Hasta ahora hemos comentado la interfaz básica de acceso a los distintos recursos. Además de ésta interfaz gráfica también es posible el uso de dos librerías muy útiles,

Formato Función	*P.Entrada*	*P.Salida*	*Descripción*
Audio			
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.

Tabla 2.15: Funciones para el audio

una utilizada para el procesamiento de imágenes, y otra que implementa un control VW en lazo cerrado para el movimiento del robot.

- 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)).

- Interfaz de movimiento VW

  El robot tiene implementado una interfaz de movimiento VW al cual se puede acceder a través de ciertas funciones. Esta interfaz está implementada como un sistema realimentado. Este sistema utiliza conjuntamente el acceso a los motores y a los encoders, de manera que se produce un sistema en lazo cerrado que posibilita un control continuo en velocidad. El controlador mantiene la velocidad constante regulando la energía que suministra a los motores. De este modo el

Formato Función	*P.Entrada*	*P.Salida*	*Descripción*
Procesado de la imagen			
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, imagendest)	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 píxel a píxel.
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).

Tabla 2.16: Funciones para el procesamiento de imágenes

control del movimiento se realiza con una precisión muy superior a la conseguida a través del movimiento de los motores de manera directa.

La librería implementa tres modos de movimiento diferentes: una traslación en línea recta de una distancia concreta a una velocidad constante, giros de un ángulo en concreto a una velocidad de giro constante, y también es posible conseguir un movimiento de giro y avance simultáneo, de manera que el robot describa un arco a una velocidad de traslación constante.

Hay que tener en cuenta que si se está utilizando el control VW y se desea realizar lecturas de los encoders será necesario hacerlas de un modo distinto al explicado en su interfaz básica.

Esto es debido a que el control VW inicializa internamente tanto los encoders como los motores, por tanto si después de inicializar el control se trata de inicializar los encoders, los manejadores devueltos no serán correctos, y por tanto las posteriores lecturas que necesiten de estos manejadores no funcionarán. De igual modo, si primero se han inicializado los encoders y después se desea inicializar el control VW éste no funcionará.

La manera de acceder a los encoders será utilizando sus funciones de la interfaz básica con los manejadores que devuelve la inicialización interna. Los valores de estos manejadores 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).

El uso de esta librería requiere de una inicialización a través de la función VWInit

(nombre_en_HDT,escala_actualización). Una vez inicializado permite realizar distintos movimientos: línea recta (VWDriveStraight (interfaz,distancia,velocidad), giro (VWDriveTurn (interfaz,radianes,velocidad)), o curva (VWDriveCurve (interfaz,distancia,radianes,velocidad_angular,velocidad_lineal)).

En la finalización del programa será necesario liberar el recurso utilizando la función VWRelease(interfaz).

2.3.7 Otros recursos

- 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).

- Acceso al conversor analógico digital
  El acceso al conversor AD permite tanto capturar un valor del mismo(OSGetAD(canal)) como activar y desactivar el conversor(OSOffAD(modo)).

- Acceso a meta información del sistema operativo
  Se pueden realizar consultas acerca de valores que conciernen al sistema, como por ejemplo, el número de versión del mismo, 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).

- Descarga de programas

  Las descargas de programas desde el PC al Eyebot se realizan a través del puerto serie. La función OSDownload se encarga de esta descarga. A esta función hay que pasarle una serie de parámetros, entre otros el nombre del programa, los bytes que se van a transmitir, la velocidad a la que se va a efectuar la transmisión, etc.

  La llamada a esta función debe de hacerse en un bucle, realizando llamadas recursivas mientras la función devuelva el valor 0, que significa que no hay ningún

Formato Función	*P.Entrada*	*P.Salida*	*Descripción*
Interfaz VW			
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).

Tabla 2.17: Funciones de la interfaz VW

error pero la transmisión no ha finalizado, por lo que hay que realizar otra recursión. En el momento en que el valor retornado sea distinto de 0 la transmisión ha terminado.

Formato Función	*P.Entrada*	*P.Salida*	*Descripción*
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 "delta"con velocidad "v". Cualquier llamada a VWDriveStraight, -Turn, -Curve o VWSetSpeed interrumpirá la ejecución de este comando.
int VWDriveTurn (VWHandle handle, radians delta, radPerSec w)	(handle) Manejador VW-Driver (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 VW-DriveStraight, -Turn, -Curve o VWSetSpeed interrumpirá la ejecución de este comando.
int VWDriveCurve (VWHandle handle, meter delta_l, radians delta_phi, meterpersec v)	(handle) Manejador VW-Driver (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_phi"y velocidad "v". Cualquier llamada a VWDriveStraight, -Turn, -Curve o VWSetSpeed interrumpirá la ejecución de este comando.
float VWDriveRemain (VWHandle handle)	(handle) Manejador VW-Driver	0.0 = si el comando VW-Drive previo ha sido completado. Cualquier otro valor = distancia que le queda por recorrer.	Devuelve la distancia que le falta por recorrer fijadas mediante VW-DriveStraight, -Turn (para -Curve sólo devuelve "delta_l")
int VWDriveDone (VWHandle handle)	(handle) Manejador VW-Driver.	-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 VW-Driver.	-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 VW-Driver.	-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.

Tabla 2.18: Funciones de la interfaz VW (II)

2.4 Entorno de programación

Antes de hacer 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

Formato Función	*P.Entrada*	*P.Salida*	*Descripción*
Latches (Buffers de I/O de bajo nivel)			
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

Tabla 2.19: Funciones para el acceso a los buffers de I/O de bajo nivel

Formato Función	*P.Entrada*	*P.Salida*	*Descripción*
Conversor Analógico Digital			
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.

Tabla 2.20: Funciones de acceso al conversor AD

librerías y cabeceras que proporciona el sistema operativo del Eyebot deben copiarse en el PC para que los programas que se escriban para el Eyebot se puedan compilar y enlazar convenientemente.

Cualquier programa que se escriba para el Eyebot se editará en el PC, con cualquier editor de texto (por ejemplo vi, emacs o xemacs). Se utiliza lenguaje ansi-C, por su versatilidad y porque se dispone de un compilador cruzado para la plataforma Motorola-68k del Eyebot. Una vez escrito el código fuente del programa, éste se compilará 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 archivo *ejecutable.hex*. Una vez obtenido el ejecutable se descarga en el Eyebot a través del puerto serie.

Formato Función	*P.Entrada*	*P.Salida*	*Descripción*
Funciones de sistema			
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.		
Tiempos			
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.

Tabla 2.21: Funciones de acceso al sistema operativo

2.4.1 Preparación del PC

Instalación 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

Formato Función	*P.Entrada*	*P.Salida*	*Descripción*
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.

Tabla 2.22: Funciones de acceso al sistema operativo (II)

Formato Función	*P.Entrada*	*P.Salida*	*Descripción*
Descarga de programas			
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.

Tabla 2.23: Función para la descarga de programas el Eyebot

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 se copiará 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 un usuario, por ejemplo el usuario prueba crea el directorio */users/prueba/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 la configuración particular de

cada PC. Dentro de *directorioeyebot/bin* debe revisarse y modificarse el fichero gcc68. También hay que 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 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 queden como */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 los ficheros *.bashrc* y *.bash_profile* que están en el directorio HOME de cada usuario y añadir la siguiente línea:

   PATH=$PATH:/directorioeyebot/bin.

   Hay que recordar que para que los cambios tengan efecto hay que reiniciar la sesión.

Copia del sistema operativo del robot en el PC Una vez que se tiene el compilador cruzado instalado es necesario copiar el sistema operativo del Eyebot en el PC. Para ello lo que hay que hacer es:

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*.

Preparación del puerto serie Para descargar programas al Eyebot entre otras posibilidades se utiliza el script *dl*. Este programa permite descargar un fichero 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. En el ejemplo anterior se tiene el COM1 en el dispositivo */dev/ttyS0*, y se decide 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 baudios son los valores posibles). También hay que comprobar (1ª linea del script) que *dl* está configurado para la shell que se está utilizando y que se tiene habilitado el permiso de ejecución.

2.4.2 Edición, compilación y descarga de programas

Enlazando las librerías en C que se han descrito en la sección anterior con el código del usuario es posible generar programas capaces de ser ejecutados en el Eyebot [5]. El siguiente programa es un sencillo ejemplo que escribe "¡¡Hola!!"en la pantalla y espera a que se pulse una tecla, tras lo cual indica por pantalla la tecla que se ha pulsado.

```
#include "eyebot.h"
#include <stdio.h>"
void main (){
int k;
LCDPrintf("¡¡Hola!!\n");
k = KEYGet();
LCDPrintf("Se ha pulsado la tecla % d \n", k);
}
```

Una vez que se tiene completo el código fuente en C (*demo.c*), es necesario compilarlo utilizando el correspondiente compilador cruzado. Para ello se puede utilizar el script *gcc68*.

- gcc68 demo.c

El resultado de la compilación es un fichero *demo.hex* apto para ser ejecutado por el microprocesador del Eyebot.

Este fichero deberá ser cargado al Eyebot, para lo cual un cable serie debe conectar el puerto serie del Eyebot con el del ordenador. Dentro de la consola del Eyebot se deberá de preparar al robot para la recepción de programas tal y como se explicó en la sección 3 de éste capítulo. Una vez que el robot esté esperando la descarga se procede a ejecutar los pasos en el PC.

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 y 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 envío sin comprimir es válido, por lo tanto pueden ser omitidas. Hay que remarcar que sólo hay que comentarlas cuando provocan la descarga incorrecta, ya que si funcionan bien es mejor tenerlas descomentadas porque la descarga del programa es mucho más rápida.

Capítulo 3

El servidor en el Eyebot: Emovil

Una vez que se ha visto cómo es el Eyebot, los sensores y actuadores de los que dispone y las distintas maneras de acceder a ellos, hay que abordar la implementación del teleoperador.

En el Eyebot es donde se produce el acceso a los recursos del robot, si este acceso se ha de llevar a cabo desde otra máquina (el ordenador externo, en nuestro caso un PC-Linux) se hace necesaria la implementación de una arquitectura cliente-servidor. En esta arquitectura el cliente (programa que se ejecuta en el PC) debe conectarse al servidor (programa que se ejecuta en el Eyebot) y establecer una comunicación con él de manera que pueda acceder de manera remota a los recursos del robot.

Para que este diálogo sea posible entre ambos programas (el cliente y servidor) se ha definido un protocolo de comunicaciones entre ambos. El protocolo define los tipos de mensajes posibles y su formato de manera que puedan ser reconocidos y/o generados por ambos programas.

En este capítulo se describirán en detalle tanto este protocolo de comunicaciones como la implementación multihebra del programa servidor.

3.1 Protocolo de comunicaciones

Las comunicaciones entre el cliente y el servidor se establecen a través de un cable serie, la velocidad de la transmisión está condicionada por la máxima velocidad del puerto RS-232 del Eyebot que son 115200 baudios.

Al tratarse de una comunicación a través del puerto serie sólo hay dos partes involucradas, por lo que no se necesita la identificación del emisor ni del destinatario de los mensajes.

En el protocolo creado para las comunicaciones entre los programas del teleoperador

no hay redundancia, es decir, los mensajes no se repiten, y en caso de que ocurra un error en la comunicación el mensaje en cuestión es desechado (no hay protección frente a errores).

Es necesario regular el flujo de información que se le envía al Eyebot, ya que si no se tiene cuidado el buffer se puede saturar y originar la pérdida de mensajes.

Además de poder emitir y recibir mensajes es necesario que tanto el cliente como el servidor sepan reconocerlos, generarlos e interpretarlos en su caso, de manera que puedan actuar en consecuencia.

En la línea de transmisión puede haber mensajes cortos o imágenes. Se diferencia entre transmisiones de imágenes bajo demanda, información sensorial por suscripción (encoders e infrarrojos) y la transmisión de comandos de movimiento. En todos los casos la iniciativa la lleva el programa cliente, ya que sabe lo que desea visualizar el usuario humano. Los mensajes transmiten la información de los sensores (infarrojos, encoders y cámara) en el sentido del Eyebot al PC. Los comandos de movimiento y peticiones de información sensorial se transmiten en el sentido contrario.

Figura 3.1: Transmisión de imágenes

Los tipos de mensajes están establecidos en el fichero *cabeceras.h* que tanto el cliente como el servidor referencian con un *#include* en su código. Pueden ser:

1. **De saludo / despedida:** Se trata de mensajes que se utilizan para detectar si en el canal de comunicaciones hay alguien preparado para la transmisión / recepción. Cuando un cliente se conecta al servidor debe enviar un saludo para iniciar la sesión, de no enviarlo el servidor no se enterará de su presencia. Cuando una de las dos partes desconecte enviará una despedida al otro. Si el cliente se

Figura 3.2: Suscripción a infrarrojos o encoders

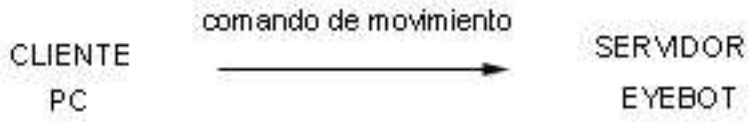


Figura 3.3: Envío de comandos de movimiento

desconecta finaliza la sesión, mientras que si el que desconecta es el servidor finalizan los programas cliente y servidor.

2. **De suscripción y des-suscripción:** Cuando se necesita recibir un tipo de información constantemente en lugar de enviar peticiones continuamente lo que se realiza es una suscripción para ahorrar mensajes en el canal. Son mensajes que se utilizan para que el servidor sepa si el cliente está suscrito o no a una serie de envíos (valor de encoders o infrarrojos) y le transmite la información sin necesidad de que haya múltiples peticiones previas cada vez que hay nuevos datos sensoriales.

3. **De datos sensoriales:** Estos mensajes envían al cliente la información a la

que en su momento se suscribió, sin necesidad de que éste vuelva a hacer una petición. Este tipo de mensajes hacen referencia al valor medido por los sensores infrarrojos o al valor de los encoders.

4. **De imagen:** Son mensajes que utiliza el cliente puntualmente para solicitar imágenes, esta solicitud tendrá una única respuesta que constará de dos partes, un mensaje de información de la imagen que se va a enviar, y la propia imagen. Una imagen se tarda relativamente poco en capturarla (se pueden alcanzar velocidades captura de 3.78 imágenes por segundo), pero la transmisión de dichas imágenes tarda mucho más, por lo que si no se tiene un control de flujo sobre este tipo de transmisiones se puede saturar el canal de comunicaciones. Por tanto un esquema de suscripción y envío automático es impensable para este tipo de transmisiones.

5. **De movimiento:** Estos mensajes se utilizan para controlar el movimiento de los distintos actuadores del Eyebot (servos y motores).

3.1.1 Formato

Los mensajes son una colección de caracteres ASCII con un formato determinado: <CABECERA> <CUERPO> \n. Este formato está pensado para facilitar el diseño de los algoritmos receptores. La composición de los mensajes cortos es diferente en función de los tipos de mensaje, pero deberá de conservar una estructura que consta de una cabecera seguida del resto del mensaje y un carácter (el "\n") que es el identificador universal de final de mensaje. La cabecera y las distintas partes en que se divide el resto del mensaje van separadas por espacios en blanco para facilitar así su identificación.

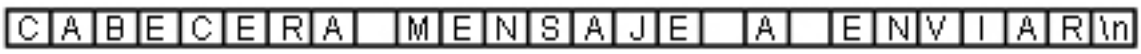


Figura 3.4: Protocolo de comunicaciones: Ejemplo de mensaje

La totalidad de mensajes posibles que se pueden transmitir aparecen en la tabla 3.1.

3.1.2 Transmisión de imágenes

Como ya se ha apuntado la transmisión de imágenes se realiza de manera distinta a la de los mensajes, ya que dentro de una imagen puede haber todo tipo de datos (incluido "\n"), por ello su trasmisión no puede hacerse igual, ya que en caso de llegar

Formato en línea de mensaje	*Descripción*	*Emisor*
PSD		
SUSCRIBEMEINFRARROJO\n	Suscripción a los PSD	cliente
BORRAMEINFRARROJO\n	Desuscripción a los PSD	cliente
TOMAINFRARROJO IZQUIERDA (valor,%d) DERECHA (valor,%d) FRENTE (valor,%d)\n	Envía el valor de los PSD	servidor
Encoders		
SUSCRIBEMEENCODER\n	Suscripción a los encoders	cliente
BORRAMEENCODER\n	Desuscripción a los encoders	cliente
TOMAENCODER IZQUIERDA (valor,%d) DERECHA (valor,%d)\n	Envía el valor de los encoders	servidor
Cámara		
DAMEIMAGENGRISASCII\n	Petición de imagen gris en modo ASCII	cliente
DAMEIMAGENGRISBIN\n	Petición de imagen gris en modo binario	cliente
DAMEIMAGENCOLORSASCII\n	Petición de imagen color en ASCII	cliente
DAMEIMAGENCOLORBIN\n	Petición de imagen color en modo binario	cliente
TOMAIMAGENGRISASCII (filas,%d) (columnas,%d) (bytes por píxel,%d) (valor máximo,%d)\n	Envío de imagen gris en modo ASCII	cliente
TOMAIMAGENGRISBIN (filas,%d) (columnas,%d) (bytes por píxel,%d) (valor máximo,%d)\n	Envío de imagen gris en modo binario	cliente
TOMAIMAGENCOLORSASCII (filas,%d) (columnas,%d) (bytes por píxel,%d) (valor máximo,%d)\n	Envío de imagen color en ASCII	cliente
TOMAIMAGENCOLORBIN (filas,%d) (columnas,%d) (bytes por píxel,%d) (valor máximo,%d)\n	Envío de imagen color en modo binario	cliente
Movimiento Motores		
COMMANDED DRIVE_SPEED (valor,%f) STEER_SPEED (valor,%f)\n	Envío de la traslación lineal y la rotación angular	cliente
STOP\n	Parada del movimiento del Eyebot	cliente
Movimiento Servos		
SERVOKICK (ángulo,%d)\n	Ancla el servo del pateador en la posición indicada	cliente
SERVOCAM (ángulo,%d)\n	Ancla el servo de la cámara en la posición indicada	cliente
General		
HOLA\n	Mensaje de inicio	cliente
ADIOS\n	Mensaje de fin de la comunicación	cliente, servidor

Tabla 3.1: Mensajes

el carácter de finalización como parte de la imagen, ésta quedaría truncada. A la hora de transmitir una imagen, ésta va precedida de un mensaje que indica su longitud y estructura. El objeto de este mensaje es indicar cuando acaba la imagen, es decir el número de datos que se han de esperar y almacenar como parte de la misma. El receptor se queda esperando hasta completar la imagen y luego pasa a esperar nuevos mensajes cortos.

La estructura de la imagen dependerá del tipo que sea. En el Eyebot se pueden obtener imágenes tanto en color como en escala de grises. A nivel físico estas imágenes representan el nivel de color o de gris por cada píxel. Si se trata de escala de grises cada píxel estará representado por un valor que oscilará entre 0 y el valor máximo elegido

(normalmente 255 o 15). Si la imagen es en color cada píxel tendrá tres valores (entre 0 y 255) que representan las componentes roja, verde y azul (RGB) del mismo.

Además los valores de los pixels se pueden transmitir en modo ASCII o en binario. Para las imágenes en escala de grises se envía el valor de cada píxel. Si este envío se realiza en modo ASCII, los valores se enviaran carácter a carácter (por ejemplo para enviar un 156 primero se enviaría un 1 luego un 5 y después un 6). Cada píxel va separado de los demás por un "\n" de manera que es posible llevar un control de los píxels transmitidos. Si las imágenes son enviadas en modo binario se enviara el valor de cada píxel como un solo carácter (por ejemplo, se enviaría el byte 10011100). De esta manera el número de píxels transmitidos es igual al número de caracteres.

En el caso de imágenes a color hay que tener en cuenta que cada píxel tiene tres componentes de color (RGB), por lo que en el modo ASCII habrá que separarlas entre sí con espacios en blanco. Igual que en escala de grises los píxels van delimitados por el carácter "\n". En el caso de envíos binarios, por cada píxel se envían tres caracteres (el valor de la componente convertido a binario). El control de los píxels enviados se realiza dividiendo por tres el número de caracteres transmitidos.

Los tipos de imágenes que se pueden transmitir se corresponden con los estandares PPM y PGM, cuyas características se describen a continuación.

Mapa de grises PGM Este tipo de imágenes se han de almacenar con la extensión *.pgm* (portable graymap file format). El formato de su contenido ha de ser el siguiente.

- Un indicador para identificar el tipo de fichero. P2 para imágenes ASCII y P5 para imágenes binarias.
- Espacio en blanco o retorno de carro.
- El ancho de la imagen (número de columnas), en caracteres ASCII.
- Espacio en blanco o retorno de carro.
- El alto de la imagen (número de filas), también en caracteres ASCII.
- Espacio en blanco o retorno de carro.
- El valor máximo en la escala de grises, en ASCII.
- Espacio en blanco o retorno de carro.
- Valores de gris para cada píxel, empezando en la esquina superior izquierda y avanzando de izquierda a derecha y de arriba a abajo. Si es una imagen ASCII van separados por espacios en blanco o retornos de carro, si se trata de una imagen en binario van uno detrás de otro.

- Las líneas que comienzan por # se consideran comentarios.

```
# ejemplo imagen.pgm
P2
7 2
15
0 0 5 9 12 15 15
0 3 4 8 10 13 14
```

Mapa de píxels PPM La extensión para almacenar estos ficheros es *.ppm* (portable pixmap format). Su definición es la que sigue.

- Un indicador del tipo de fichero. P3 para imágenes ASCII y P6 para las binarias.
- Espacio en blanco o retorno de carro.
- El ancho de la imagen (número de columnas), en caracteres ASCII.
- Espacio en blanco o retorno de carro.
- El alto de la imagen (número de filas), también en caracteres ASCII.
- Espacio en blanco o retorno de carro.
- El valor máximo que pueda tomar el valor de una componente de color..
- Espacio en blanco o retorno de carro.
- Los valores de las componentes RGB de cada píxel, separadas entre ellas por espacios en blanco si se trata de imágenes ASCII o sin separación para imágenes ASCII. Se empieza por el píxel situado en la esquina superior izquierda y se avanza de izquierda a derecha y de arriba a abajo.
- Las líneas que comienzan por # se consideran comentarios.

```
# ejemplo imagen.ppm
P3
4 4
255
 10  9   1      25  30  19      90 15   2      15  210 190
135 97 133     255 100  90      12 55   8     255 200 180
200  0  48      35  40  73      85 23 108     161 222  90
193 14  28     115  33  62      90 93  88     135 102  61
```

Para ambos tipos el valor máximo que puede alcanzar un píxel o su componente para las imágenes binarias es de 255, mientras que para imágenes ASCII no está delimitado, usándose frecuentemente 255 o 15.

En cualquier caso hay que destacar que las imágenes binarias ocupan considerablemente menos, por lo que su transmisión es más rápida (hasta aproximadamente tres veces más que para imágenes a color). Este modo es el que se usa por defecto en la implementación final.

3.2 Implementación multihebra del programa servidor

El programa servidor se encarga de detectar el establecimiento y el cierre de las sesiones con el cliente, de enviarle la información que aquel le solicite y de ejecutar los comandos que el cliente le envíe. Consta de una fase de inicialización y está articulado en tres hebras de funcionamiento contínuo y simultáneo.

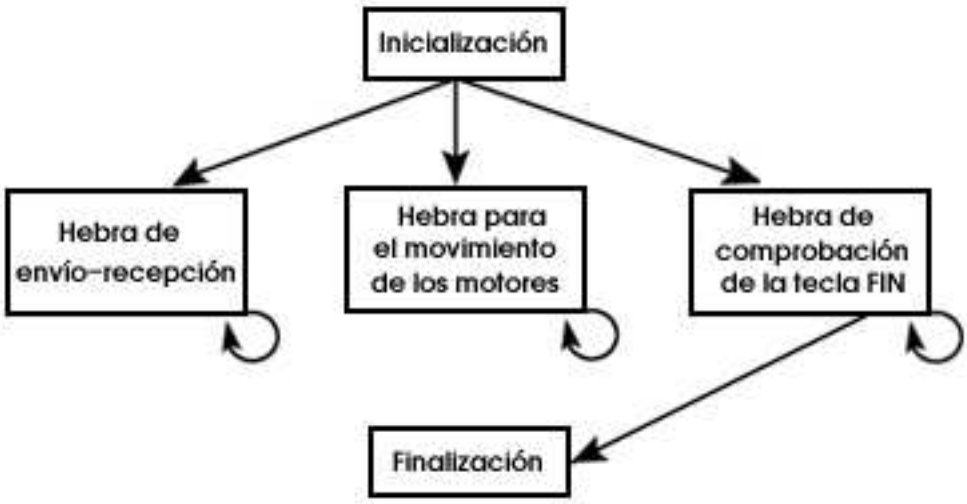


Figura 3.5: Organización del programa servidor

Antes de realizar cualquier acción el programa necesita inicializar todo lo necesario para posteriormente poder acceder a los distintos dispositivos. En este sentido la fase

de inicialización se encarga de poner a punto los siguientes elementos:

1. Puerto serie: De los puertos posibles del Eyebot se utiliza el SERIAL1. Este puerto se configura a una velocidad de 115200 baudios. Se elige esta velocidad, la máxima, porque es necesaria para transmitir imágenes, de lo contrario el sistema resultaría excesivamente lento.

2. Control VW: Para posibilitar un movimiento controlado y preciso del robot es necesario inicialización el control VW. Esta librería utiliza un controlador proporcional integral (PI) que se activa con las distintas llamadas a la librería. El controlador PI es el encargado de la realimentación del sistema, de manera que tratará de mantener una velocidad constante en los movimientos adaptando la energía que se suministre a los motores.

   Para su inicialización los parámetros necesarios han sido elegidos heurísticamente, resultando los valores de 5 y 0.1 para los valores de las componentes proporcional e integral respectivamente del controlador en velocidad de traslación. Para el controlador en velocidad de giro el valor de la componente proporcional de la velocidad de giro es 5, y 0.1 el de la componente integral.

   Al inicializar este control los encoders se preparan internamente, por lo que no hay que inicializarlos explícitamente.

3. Servos: Se inicializa tanto el servo encargado del movimiento de la cámara como el encargado del movimiento del pateador, asignándoles a cada uno un manejador de manera que sea posible posteriormente moverlos.

4. Infrarrojos: Se inicializan los tres y se les asigna un manejador para posteriormente acceder a ellos.

5. Modo multitarea: Se elige inicializar la multitarea en el modo con desalojo, que como ya se ha comentado permite que se ejecuten las distintas tareas con independencia de que alguna esté bloqueada.

Como se observa en esta lista, la cámara no se inicializa en estos momentos. Esto es debido a que por una limitación del sistema las interrupciones originadas por la cámara bloquean la recepción por el puerto serie, por lo que se ha hecho necesario inicializar la cámara únicamente cuando se desee capturar una imagen, tras lo cual se libera el recurso.

Tras esta inicialización global de recursos se crean tres hebras que se ejecutan en paralelo, cada una de ellas en un bucle infinito. Es necesario el uso de hebras que se ejecuten en paralelo porque se ha distribuido el programa en tres tareas relativamente independientes, ya que por la naturaleza del programa es más facil este desarrollo que uno secuencial. Se desea que las tareas sean independientes entre sí, ya que los motores se han de seguir moviendo aunque se esté recibiendo una imagen, y de igual manera se ha de comprobar si se ha pulsado la tecla de finalización independientemente de si se mueven los motores.

Dentro del sistema multitarea una hebra se encarga del envío y recepción de datos, así como de componer mensajes, analizarlos y darles servicio. Otra hebra se encarga del movimiento de los motores y una última hebra es la responsable de comprobar constantemente si se ha pulsado la tecla de finalización en el Eyebot.

La hebra encargada del envío y recepción de datos se inicializa con mayor prioridad, ya que es la más importante, mientras que las otras dos tienen una prioridad inferior. Una vez inicializadas se insertan en el planificador y se preparan para empezar a ser ejecutadas. Una vez terminada la fase inicial de configuración se activa el planificador que se hace cargo del flujo de ejecución, que pasa a las distintas hebras mencionadas.

El programa sólo retornará a la función principal después de haber detenido todas las tareas. En el momento de retornar se liberarán todos los recursos que se hayan ocupado, se detendrán los motores si se estuvieran moviendo y se enviará un mensaje de ADIOS para que en el caso de que hubiera algún cliente escuchando éste se desconecte.

3.2.1 Hebra de envío y recepción

Esta hebra es llamada por el planificador con la máxima prioridad, entrando en un bucle infinito dentro del cual realiza consecutivas llamadas a la rutina encargada de recibir datos por el puerto serie y a la rutina encargada de dar servicio a las suscripciones.

Además de realizar lecturas del puerto serie esta hebra también envía periódicamente nuevos datos sensoriales, por tanto se desea que las lecturas del puerto serie se hagan en modo no bloqueante. De este modo no se detiene el flujo de la hebra aunque no haya nuevos datos para leer. Si se realizara una lectura bloqueante se detendría en este punto hasta que hubiera algo que leer. Cuando hay datos para leer se procede a su lectura a través de la función del sistema operativo OSRecvRS232 (carácter, interfaz), lo que implica que sólo se podrán hacer lecturas de carácter en carácter.

Para compensar esta limitación en cada iteración de esta hebra se realiza una lectura

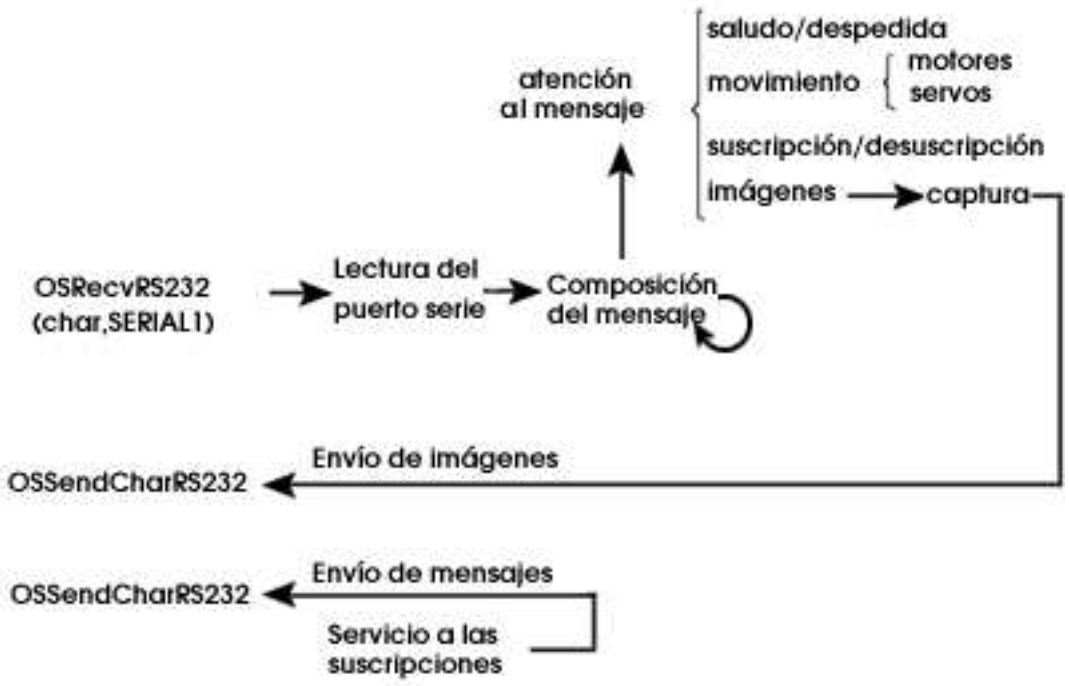


Figura 3.6: Hebra de envío y recepción

en modo exhaustivo, es decir, se harán sucesivas lecturas hasta que no haya nada en el puerto. Después de cada lectura se analiza el carácter que se ha recibido. Si no se trata de un carácter "\n " la lectura continúa, pero en caso contrario significa que hay un mensaje que ha sido completado, por lo que se pasa a analizarlo haciendo una llamada a la rutina *"procesa_mensaje"*. En esta rutina se trocea el mensaje por el primer espacio en blanco para extraer la cabecera. Sabiendo la cabecera se conoce el resto de componentes del mensaje que se obtienen separando por los espacios en blanco. Una vez extraída la cabecera se compara entre las distitnas posibilidades.

1. Si la cabecera es un saludo (*HOLA*) se abre la sesión. A partir de ese momento los mensajes que lleguen son escuchados y atendidos.

2. Si se trata de una cabecera de despedida (*ADIOS*) se cierra la sesión, y se detienen los motores en caso de que se estuvieran moviendo. A partir de ese momento se ignorarán los mensajes que lleguen mientras no se reciba otro mensaje de saludo.

3. Si se recibe una cabecera *STOP* los motores serán detenidos.

4. *SUSCRIBEMEENCODER* activa la suscripción del cliente a los encoders, mientras que *BORRAMEENCODER* la desactiva.

5. Igualmente *SUSCRIBEMEINFRARROJO* activa la suscripción del cliente a los infrarrojos, mientras que *BORRAMEINFRARROJO* la desactiva.

6. La cabecera *COMMANDED* indica que en el resto del mensaje hay instrucciones de cómo mover los motores y se activa el flag que permite el movimiento del Eyebot.

7. En caso de tratarse de una cabecera *SERVOCAM* o *SERVOKICK* se indica el anclaje de los servos en una posición, por tanto se actúa sobre el servo convenientemente.

8. Si la cabecera es *DAMEIMAGENGRISASCII* o *DAMEIMAGENGRISBIN* o *DAMEIMAGENCOLORASCII* o *DAMEIMAGENCOLORBIN* el cliente esta solicitando una imagen, por lo que se captura el tipo de imagen que se haya solicitado y se devuelve un mensaje del tipo *TOMAIMAGENGRISASCII* o *TOMAIMAGENGRISBIN* o *TOMAIMAGENCOLORASCII* o *TOMAIMAGENCOLORBIN* según corresponda, indicando las características de la imagen que se va a enviar. Finalmente se envía efectivamente la imagen correspondiente con las funciones *"sendGreyascii"*, *"sendGreyRaw"*, *"sendRGBascii"* y *"sendRGBRaw"* respectivamente.

Las cuatro rutinas que se encargan del envío de imágenes se diferencian entre sí por el tipo de imagen que transmiten (color o escala de grises) y el modo en el que lo hacen (binario o ASCII).

Además de componer los mensajes que se reciben esta hebra se encarga de enviar constantemente al cliente los datos sensoriales necesarios. La hebra comprueba si hay alguna suscripción a encoders o infrarrojos que deba ser atendida, y si la hay ejecuta las rutinas de envío de los datos correspondientes. Estas rutinas son *enviapsd* para la suscripción a infrarrojos y *enviaencoders* para la suscripción a encoders. En ellas primero se lee los valores de los sensores pertinentes y después se componen los correspondientes mensajes (*TOMAINFRARROJO IZQUIERDA valor DERECHA valor FRENTE valor*\n para los infrarrojos y *TOMAENCODER IZQUIERDA valor DERECHA valor*\n para los encoders). Una vez compuesto el mensaje, lo envía por el puerto. El envío se efectúa a través de la rutina *"sendstr"*, que envía estos mensajes

carácter a carácter a través de la función del sistema operativo OSSendCharRS232 (carácter, puerto).

Figura 3.7: Servicio a las suscripciones

3.2.2 Hebra para el movimiento de los motores

Formando parte de las tareas del planificador e inicializada con una mínima prioridad se encuentra la hebra responsable del movimiento de los motores. Esta hebra ejecuta un bucle infinito en el que se comprueba si hay que mover los motores (con anterioridad se ha recibido una cabecera del tipo *COMMANDED*). En caso afirmativo lee comandos de velocidad de traslación y ángulo de giro que ha recibido y llama a las funciones del control VW para mover efectivamente el robot.

Si la velocidad de traslación es inferior a 0.01 m/s no se realiza un movimiento que implique un desplazamiento del robot, ya que con esa velocidad fijada los motores no pueden con el peso del robot, por tanto se efectúa un giro del robot sobre sí mismo utilizando la función VWDriveTurn (manejador_del_control, giro, velocidad). Mientras el cliente no envíe otro comando de movimiento o se ordene la parada en cada iteración de la hebra se ordena el giro, resultando el efecto de que el robot rota sobre sí mismo todo el tiempo. El sentido del giro vendrá determinado en el mensaje recibido, siendo un giro a izquierdas para valores positivos y a derechas para valores negativos.

Si la velocidad de traslación es mayor que 0.01 m/s y el giro es inferior a $\frac{\pi}{6}$ radianes el robot se desplazará en línea recta a velocidad constante e igual al valor indicado en el mensaje recibido para efectuar el movimiento. La función de la librería VW que permite realizar este movimiento es VWDriveStraight(manejador_del_control, metros, velocidad). Con la llamada a esta función aisladamente se consigue que el robot se desplace un número determinado de metros a una velocidad constante, tras lo cual se detiene. Sin embargo en nuestro contexto para conseguir un control en velocidad se aprovecha el hecho de que realizar una llamada a una función de movimiento del control VW anula la función de movimiento que se estuviera ejecutando con anterioridad. Si se pone un valor de desplazamiento en metros lo suficientemente grande (por ejemplo 3m), el movimiento no habrá acabado cuando se realice la siguiente iteración de la hebra. En la nueva iteración se vuelve a llamar a la función para que vuelva a avanzar desde la posición actual (*efecto zanahoria*). De este modo se consigue un control en velocidad de desplazamiento.

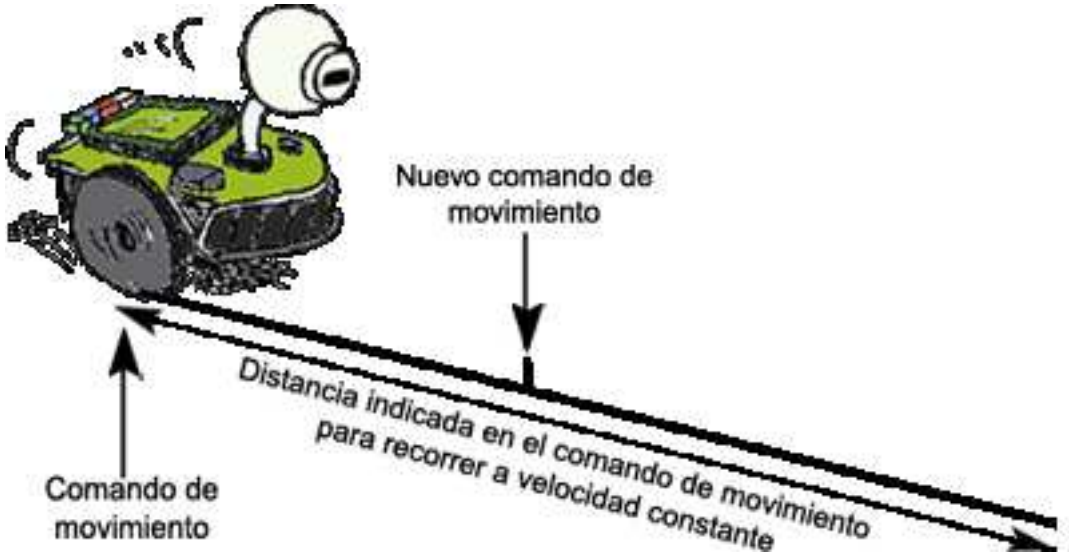


Figura 3.8: Velocidad constante con sucesivas llamadas a desplazamientos.

El sentido de la traslación viene determinado por el signo de la velocidad, este signo se pasa en el signo de la traslación en metros solicitada (positivo avanza, negativo retrocede), teniendo en cuenta que para ordenar el movimiento la velocidad tiene que ser siempre positiva.

Aprovechando el mismo planteamiento para la consecución de un desplazamiento continuo a velocidad constante, cuando la velocidad indicada es mayor 0.01 m/s y el ángulo de giro es superior a $\frac{\pi}{6}$ radianes el robot describe un arco utilizando la función

VWDriveCurve(manejador_del_control, metros, ángulo, velocidad) con un control en velocidad de traslación y en el ángulo del arco recorrido. A la función es necesario indicarle una distancia (longitud del arco a recorrer) lo suficientemente grande (en nuestro caso 3m) para que el movimiento no acabe en una sola iteración, junto con el ángulo al que se corresponde y la velocidad a la que se desea recorrerlo.

El ángulo del arco y la velocidad a la que hay que recorrerlo son indicados en el mensaje. El signo del ángulo determina si el arco se recorre hacia la izquierda (positivo) o hacia la derecha (negativo). El signo de la velocidad indica si el desplazamiento será hacia delante o hacia atrás, debiendo tener en cuenta que la velocidad que se pase como parámetro a la función debe ser siempre positiva.

Si cuando se está ejecutando un movimiento se reciben instrucciones de realizar otro distinto el movimiento anterior queda anulado inmediatamente, al igual que si se recibe un mensaje que indique la detención del robot.

3.2.3 Hebra para el control de la tecla de FIN

Esta hebra se encarga de comprobar en un bucle infinito si se ha pulsado la tecla de finalización en el Eyebot. De ser así concluye todas las tareas, por lo que el flujo de ejecución retorna a la función principal donde finaliza todo el sistema.

Figura 3.9: Comprobación de la tecla FIN

Capítulo 4

Herramientas para la programación en el PC

El programa cliente es el encargado de establecer el inicio de sesión con el Eyebot, de solicitarle información sensorial y de enviarle comandos de movimiento. La información sensorial que recibe la debe de representar en una interfaz gráfica para que el usuario la pueda visualizar. Desde esta interfaz el usuario también puede mover el robot. El entorno en el que va a ejecutarse este programa es el PC.

Para conectarse con el programa servidor que se está ejecutando en el Eyebot el cliente debe de hacer uso del puerto serie RS-232. Este puerto puede ser configurado desde programa utilizando las funciones *ioctl* (Input/Output Control). Una vez configurado el puerto se puede acceder a él utilizando las distintas llamadas al sistema: *read*, *write*, *open*, *close*, etc.

El cliente presenta una interfaz gráfica al usuario, debiendo interactuar con él. Desde esta interfaz se permite al usuario acceder a los actuadores del robot de manera sencilla, y a la vez le muestra las medidas de los distintos sensores en pantalla. Para construir dicha interfaz se ha utilizado la librería XForms. Esta herramienta permite de manera sencilla una construcción gráfica de la interfaz y contiene los elementos visuales necesarios (*joystick*, botones, posicionadores...) para las representaciones que se desean.

Además de encargarse de las comunicaciones con el Eyebot, el programa cliente deberá ejecutar otras acciones, como comprobar la interacción del usuario con la interfaz gráfica, refrescar convenientemente esta interfaz, etc. Estas acciones han de ejecutarse a la vez. El programa cliente se ha implementado como un conjunto de tareas que se describen en detalle en el capítulo 5. Frente a otras opciones como Pthreads se ha preferido utilizar el planificador Rai-Scheduler [11] para implementar este paralelismo,

por ser una librería utilizada anteriormente por el grupo de robótica.

En este capítulo se verán tanto la manera de inicializar y acceder al puerto serie desde el programa del usuario, como la funcionalidad y utilización de las librerías Rai-Scheduler y XForms.

4.1 Programación del puerto serie

A través del puerto serie se realiza la descarga de los programas usuarios desde el PC al Eyebot, pero este puerto también puede ser utilizado para la comunicación entre el programa que se está ejecutando en el robot, y un programa que se ejecuta en el PC. Precisamente se usará de este modo en este proyecto.

El programa cliente dispone de la funcionalidad que le ofrece el *kernel* de Linux para acceder al puerto serie en forma de llamadas al sistema [2]. La configuración de las características del puerto serie se realiza con las llamadas *open* (que abre el puerto serie), *ioctl* y *fcntl*. La lectura se hace con la llamada *read*, mientras que para la escritura se utiliza con *write*. El puerto se cierra con la llamada *close*.

La llamada al sistema *open* se utiliza para convertir un nombre (en este caso el del dispositivo del puerto serie, normalmente */dev/ttyS0* o */dev/ttyS1*) en un descriptor de fichero (un entero positivo) que se utiliza para las operaciones de entrada / salida posteriores. En caso de error el valor devuelto es <= 0. Es necesario indicar el modo en el que se va a utilizar el descriptor. Estos modos pueden ser de sólo lectura, sólo escritura o lectura y escritura. Para el sistema teleoperador el modo utilizado es el de lectura y escritura. El nuevo descriptor de fichero se puede configurar de manera que se fije la velocidad de transmisión a través del puerto serie (llamadas a la función *fcntl*), o bien se configuren otras características de la transmisión, como la paridad, el número de bits..., (funciones *ioctl*).

La lectura del puerto serie se realiza con la llamada al sistema *read*, que lee de un descriptor de fichero la cantidad de bytes que se le indiquen, guardándolos en una zona de memoria especificada en la llamada. Esta función devuelve el número de bytes que realmente se han leído, y si ha habido algún error devuelve un -1. Se utiliza para recibir los mensajes enviados por el programa servidor a través del puerto serie.

Para escribir en el puerto serie se utiliza la llamada al sistema *write*. Esta llamada escribe en el fichero referenciado el número de bytes indicados, desde el comienzo del buffer especificado en la llamada. Si todo funciona correctamente se devolverá

el número de bytes escritos, pero si hay algún error el valor devuelto será un entero negativo. Se utiliza para que el programa cliente envíe sus mensajes al programa en el Eyebot a través del puerto serie.

Para cerrar el descriptor de fichero que referencia al puerto serie se utiliza la llamada al sistema *close*. El valor devuelto es 0 si todo funciona correctamente y -1 si hay algún error.

La lectura y escritura en el puerto serie se puede hacer en modo bloqueante o no bloqueante. En modo bloqueante cuando se hace una lectura el programa se queda esperando si no hay ningún dato que leer el programa, hasta que haya alguno. De igual manera ocurre con la escritura, de modo que se espera hasta que los bytes se han escrito realmente. Por el contrario en el modo no bloqueante si no hay datos para leer o si la escritura no se ha materializado el programa continúa ejecutándose. Para un programa cliente multihebra es deseable un acceso no bloqueante, de manera que ninguna de sus hebras quede detenida al realizar una llamada de lectura o escritura en el puerto serie.

Las condiciones de utilización del puerto serie desde el PC son menos rígidas que para el robot ya que los buffers de entrada y salida son mayores. Además el PC puede leer todo lo que haya en el buffer con una sola llamada *read* (lectura exhaustiva). Todas las características del puerto serie del PC pueden ser configuradas por el programador, por lo que se han de adaptar a los valores impuestos para el Eyebot en cuanto a velocidad, bit de paridad, etc. Esto se hace con llamadas *ioctl*, que son un conjunto de llamadas que permiten el acceso y configuración del puerto serie.

4.2 Paralelismo en el PC

El programa cliente está implementado como varios hilos de ejecución. Para programar en paralelo hay varias opciones en el PC. Una de ellas es el uso de Pthreads, que en las distribuciones recientes de Linux ya viene incluida. El manejo de Pthreads no es fácil y requiere de una programación cuidadosa con el acceso a variables compartidas, ya que de lo contrario aparecen problemas de concurrencia. Dentro del grupo de robótica proyectos anteriores utilizaron la librería Rai-Scheduler [11] como alternativa al uso de Pthreads.

El Rai-Scheduler es un planificador de tareas, escrito en C, que permite al usuario programar en paralelo y pensar en las distintas tareas como distintas hebras que se

ejecutan simultáneamente dentro del procesador. En realidad no es paralelismo estricto sino pseudo-paralelismo, ya que para Linux se presentan como un único proceso, pero a efectos prácticos es equivalente. Además cuenta con la simplificación de los problemas de concurrencia. No hay condiciones de carrera entre unas tareas y otras porque nunca se ejecutan como procesos diferentes.

Rai-Scheduler implementa sistemas multitarea que funcionan en pseudo-paralelismo, siendo este planificador el encargado de distribuir entre sus tareas el tiempo de CPU que le corresponde como proceso Linux. Rai-Scheduler no implementa el desalojo, si una tarea se bloquea el resto quedan detenidas. Con la librería Pthreads esta distribución de tiempo la realiza el planificador del sistema operativo, obteniendo así programas más eficientes porque es capaz de desalojar las tareas. En cualquier caso se asume que los plazos de las tareas no son críticos ya que Linux no es un sistema de tiempo real. La decisión de utilizar Rai-Scheduler se debe principalmente a que es una librería ampliamente conocida por el grupo de robótica.

4.2.1 Rai-Scheduler

La funcionalidad de Rai-Scheduler se ofrece en forma de librería *libRai.a*, con la cabecera *rai/Rai.h*. Esta librería obliga a pensar en la aplicación del usuario como un conjunto de tareas ejecutándose en paralelo. El uso de la librería es sencillo, las tareas que se insertan al planificador pueden ser de tres tipos:

1. **Tareas periódicas** Las tareas cogen control periódicamente y se ejecutan en forma incremental, como iteraciones que son llamadas cíclicamente.

2. **Tareas evento** Estas tareas se ejecutan como respuesta a un evento. Son funciones *callback* que se llaman cuando dicho evento se genera. Éste evento es la escritura en cierto descriptor de fichero (fichero, socket, puerto serie...)

3. **Tareas temporizador** Tareas que se ejecutan cuando cierto temporizador alcanza su cuenta final,siendo posible actualizar ese temporizador.

Para que las tareas funcionen correctamente primero se ha de crear el planificador llamando a *RaiInit()*, una vez hecho se configuran sus módulos para que puedan ser insertados. Estos modulos son los que implementas las tareas. Además al configurarlos se les puede indicar la función de finalización que ejecutarán cuando la tarea sea eliminada del planificador o cuando el planificador se detenga.

En el proceso de configuración se les proporcionará información acerca de la función que han de ejecutar, el tipo de tarea que serán, su período o el evento al que están asociados. Finalmente el control del programa se cede al planificador llamando a la función *RaiStart()*. A partir de ese momento el flujo del programa sigue la activación de las distintas tareas.

Mientras el planificador esté funcionando se encargará de las llamadas a las funciones *callback* de los módulos creados en el momento que corresponda. Se puede modificar el intervalo de los *callback*, desactivarse y si lo desea puede detener el planificador haciendo una llamada a la función *ShutDown()*. Cuando ocurre, el planificador se detiene y las tareas ejecutan sus funciones de finalización. El flujo del programa es entonces devuelto a la función que invocó al planificador.

4.3 Interfaces gráficas

Un aspecto relevante de nuestras aplicaciones es su interfaz gráfica. A través de ella el robot muestra los valores de los distintos sensores de un modo gráfico. De este modo para el usuario resulta sencillo interpretar lo que se le está mostrando por pantalla. Esta interfaz gráfica también ofrece herramientas para que usuario humano pueda mover el robot a su voluntad.

X-Window es el sistema de ventanas escogido para desarrollar nuestras interfaces, por ser el más extendido dentro del mundo Unix-Linux y ajustarse a nuestras necesidades.

4.3.1 X-Window system

El sistema X-Window [14] fue desarrollado a mediados de los años 80 en el MIT, movido por la necesidad de dotar al sistema operativo Unix de una interfaz gráfica de usuario. X-Window es el encargado de visualizar la información de manera gráfica y es totalmente independiente del sistema operativo (los sistemas Unix-Linux no necesitan de X-Window para funcionar, pudiendo trabajar en modo texto).

Es el sistema gráfico más extendido en el mundo Unix-Linux. La casi totalidad de las aplicaciones gráficas desarrolladas para Linux han adoptado este sistema llegando a convertirlo en el estándar. Así pues Linux ofrece todas las librerías necesarias para desarrollar aplicaciones y generar salidas gráficas conforme a ese estándar.

La gran diferencia entre X-Window y la interfaz gráfica de otros sistemas operativos

es que X-Window distribuye el procesamiento de aplicaciones, especificando un enlace cliente-servidor. El cliente X especificará "qué hacer" al servidor X, que se encargará de "cómo hacerlo". El servidor X es un programa que se ejecuta en una máquina. A este servidor le llegan solicitudes de clientes X para generar la salida gráfica de una aplicación en la pantalla que administra el servidor, que puede ser la de la máquina que ejecuta el cliente o la de otra máquina remota.

El sistema X-Window oculta las peculiaridades del sistema operativo y del hardware asociado al mismo, simplificando la labor del desarrollador y proporcionando una alta portabilidad de las aplicaciones cliente.

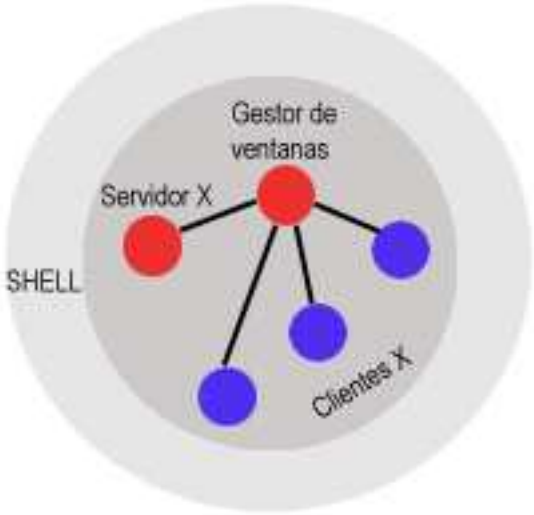


Figura 4.1: Arquitectura cliente/servidor en una máquina

La gran ventaja de X-Window es que el servidor X de una aplicación y el cliente X donde se trabaja no tienen por qué estar en la misma máquina. Se puede utilizar X-Window en una máquina y conectarse a otra remota, ejecutar un programa en esta máquina remota y visualizar e interactuar con este programa en la máquina local.

Esta característica de X-Window permite utilizar la interfaz gráfica del teleoperador en cualquier máquina que soporte este sistema gráfico, consiguiéndose así una implementación adicional a la perseguida, que es la posibilidad de hacer una *teleoperación remota* sobre el robot. Esto implica que el operario de la interfaz gráfica no ha de estar obligatoriamente delante del ordenador donde se ejecuta el programa cliente, sino que puede conectarse a éste desde cualquier ordenador y teleoperar el robot.

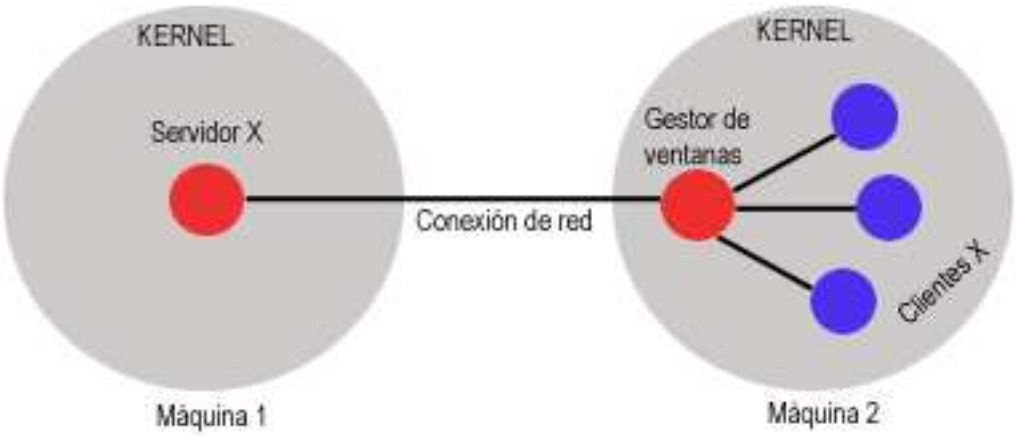


Figura 4.2: Arquitectura cliente/servidor en una red

4.3.2 Librería XForms

El sistema X-Window es un sistema relativamente complejo, que ofrece varias interfaces de acceso a su funcionalidad, desde las de más bajo nivel hasta otras más elaboradas que ocultan al programador detalles que normalmente no necesita. Las interfaces de bajo nivel permiten máxima flexibilidad y control de los detalles de la interacción gráfica, pero suelen resultar demasiado complicadas para la mayor parte de aplicaciones.

En esta dirección han surgido paquetes software que se montan encima del sistema X-Window ofreciendo distintos objetos gráficos y un repertorio de patrones de interacción y visualización. Esta reducción de la flexibilidad total del bajo nivel disminuye la complejidad de uso sin recortar excesivamente la funcionalidad que las aplicaciones normales pueden necesitar, facilitando de este modo la programación de interfaces gráficas. La librería XForms [12] es uno de estos paquetes, precisamente el que se ha elegido para desarrollar las interfaces gráficas que necesitemos. Este paquete cuenta con varias ventajas:

- Es software de libre distribución, completamente gratuito para uso no comercial (accesible en *http://bragg.phys.uwm.edu/xforms*).
- Además ofrece un repertorio extenso de elementos gráficos (botones, *diales*, *canvas*, etc.), suficiente para las necesidades que tenemos. Oculta mucha complejidad asociada a las ventanas gráficas, mostrando una interfaz de uso fácil y coherente.

- XForms está en perfecta sintonía con el entorno informático disponible. Se ejecuta sobre sistema X-Window, con lo que la visualización remota es posible, y está escrita en C.

- También ofrece una herramienta visual, llamada *fdesign* , muy sencilla, para crear la interfaz gráfica del programa aplicación.

4.3.3 Elementos gráficos de XForms

La librería ofrece un amplio repertorio de objetos gráficos con los que confeccionar el frontal gráfico según se desee, cada uno de ellos con un determinado comportamiento y funcionalidad. Estos elementos son sensibles a la interacción con el usuario a través del ratón y del teclado. Por otro lado son accesibles y manipulables desde el código del programa aplicación. Así permiten la comunicación bidireccional: al usuario comunicar cosas al programa y al programa mostrar cosas al humano, es decir, la interacción gráfica entre ambos.

Para la implementación de la interfaz gráfica del teleoperador el programa cliente necesita utilizar los siguientes elementos gráficos:

Botón que el usuario puede pulsar. De diferentes tipos según su comportamiento cuando se pulsa con el ratón: los hay que notifican el evento al pulsarlo, que lo notifican cuando se suelta, otros se quedan en estado "pulsado" hasta que se pulse otra vez y los hay que vuelven inmediatamente a su estado de reposo.

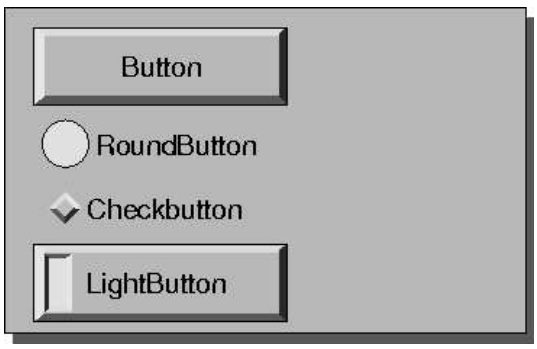


Figura 4.3: XForms: Botones

Dial (*slider* si no muestra el valor, *varslider* si lo indica) con los que el usuario puede fijar un valor dentro de un rango lineal.

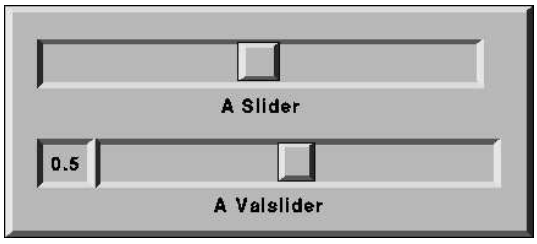


Figura 4.4: XForms: Diales

Canvas que permite al programa pintar sus propias imágenes, puntos, rectas, etc. Nuestro programa pinta en este *canvas* las líneas de los infrarrojos, la silueta del robot,... dependiendo de lo que el operario quiera visualizar. También permite volcar a pantalla imágenes reales obtenidas con la cámara.

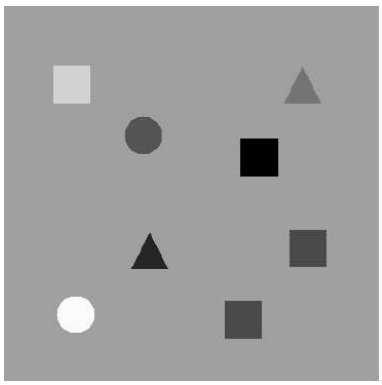


Figura 4.5: XForms: Canvas

Posicionador que permite conocer la posición x y que se ha seleccionado con el ratón gracias al movimiento de unas trazas sobre un área. De este modo se especifican 2 valores con un solo *click* de ratón. Este elemento lo utilizaremos para crear el *joystick* visual.

4.3.4 Construcción visual de la interfaz gráfica con XForms

Una vez que se tiene en mente el diseño se construye el frontal gráfico con la herramienta visual que proporciona XForms. Ésta construcción conlleva la selección de los distintos elementos gráficos que conforman la interfaz, su tamaño, su posición y sus atributos.

El propio paquete XForms ofrece una herramienta visual para construir el frontal gráfico de la aplicación. Esta herramienta se llama *fdesign* y crea una ventana vacía sobre la cual permite ir incorporando distintos elementos gráficos del repertorio (botones,

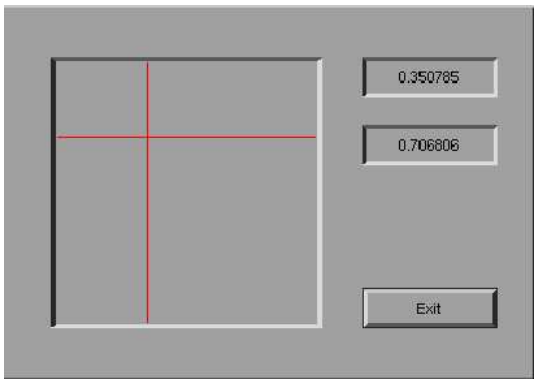


Figura 4.6: XForms: Posicionador

diales, cuadros de texto, *canvas*...), que se van situando rellenando el frontal gráfico. *Fdesign* también permite configurar estos elementos: color, tipo de letra, aspecto, color cuando el ratón lo pulsa, tamaño, forma, etc.

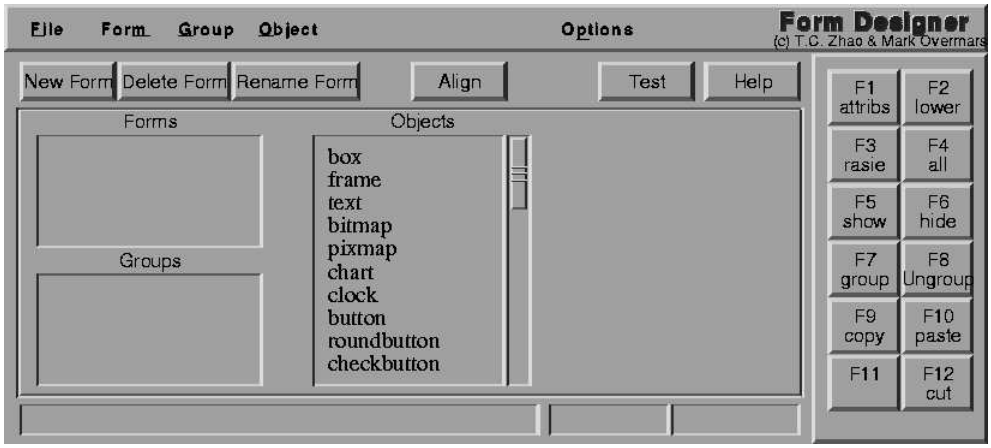


Figura 4.7: Fdesing

La posición se decide con el ratón, seleccionando dentro del repertorio el elemento que queremos insertar, arrastrando hasta donde queremos situarlo en el frontal y soltándolo allí. El tamaño también se varía visualmente con el ratón, del mismo modo que el tamaño de las ventanas de X-Window. Otro tipo de propiedades como color, etiquetas, etc. se configuran cambiando las propiedades del objeto, también visualmente en gran medida, pulsando entre las distintas posibilidades.

Una vez acabado el diseño del frontal la herramienta genera tres ficheros: *nombre.fd* en el que guarda todos los elementos con sus atributos y los ficheros *nombre.h* y *nombre.c* que son los equivalentes en lenguaje C. Gracias a éstos últimos los programas del

usuario pueden iniciar la visualización del frontal, acceder a los distintos elementos del mismo y manipularlos a voluntad. Los objetos gráficos tienen asociadas gran cantidad de funciones que permiten muestrear y fijar su estado, acceder y cambiar sus valores, hacer un repintado, etc.

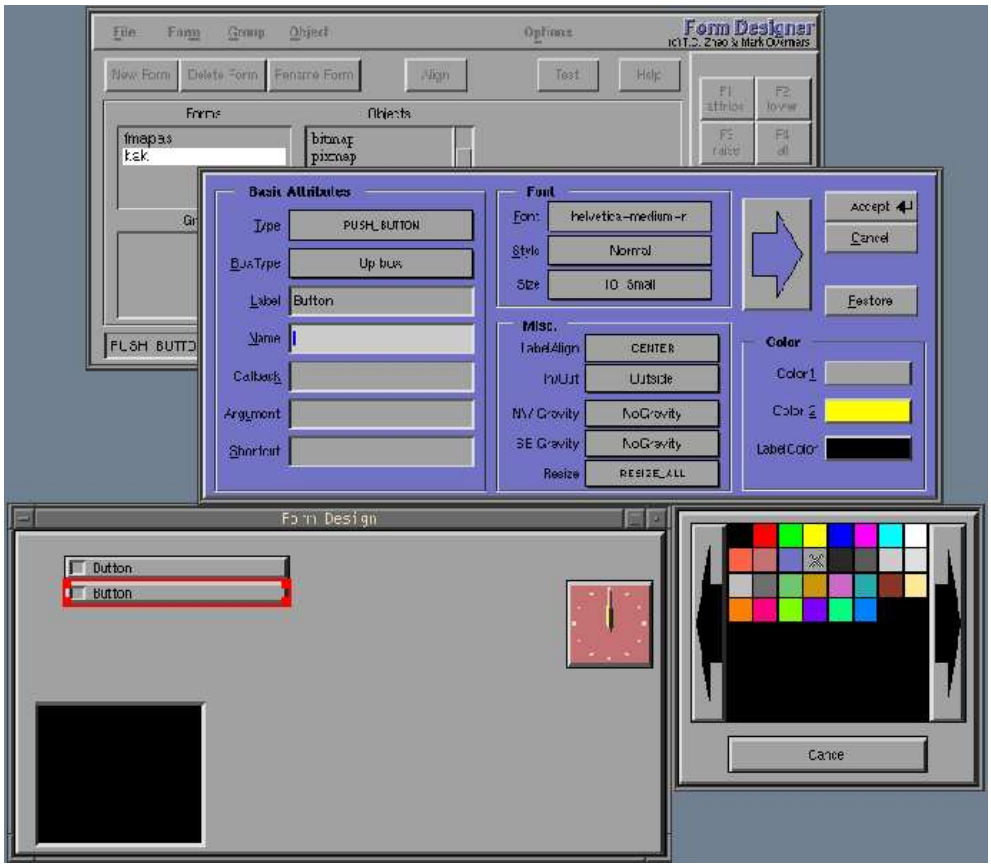


Figura 4.8: Construcción de una interfaz gráfica con Fdesign

4.3.5 Interacción de XForms con el programa

La librería XForms ofrece un modelo de interacción basado en eventos. Se asocia un área de la ventana a los elementos gráficos y se genera eventos cuando el ratón pasa por encima, cuando el ratón se pulsa o se suelta en ese área, etc. El paquete permite definir *callbacks*, que son funciones enganchadas a cierto evento sobre un objeto. Estas funciones son invocadas cuando ese evento se produce en el objeto asociado. El enganche entre el evento y la función se realiza en el momento de construir la interfaz gráfica, el programador no debe vigilar de nada, salvo definir esa función, la librería se encargará de llamar al *callback* cuando proceda.

La librería XForms permite dos modos de operación. En el primero toma completamente el control del flujo de ejecución y deja todo el funcionamiento del programa en modo *reactivo*, respondiendo a los eventos que el usuario genera. El segundo esquema permite *muestrear*, de modo no bloqueante, el estado de la interfaz y mantener el control del flujo de ejecución en el programa de la aplicación, que decide cuándo muestrear y cómo responder al estado en que se encuentre la interfaz [10]. Éste es el esquema que hemos seguido en nuestra aplicación, con una hebra periódica que muestrea el estado del frontal 4 veces cada segundo.

El muestreo de los elementos del frontal recorre los distintos objetos de los que consta. La función *fl_check_forms()* devuelve el elemento que ha sido modificado refiriéndose a él como *fd_nombredelForm -> elemento*. Además de saber el elemento que ha sido modificado, también es posible saber el estado en el que se encuentra y, si se desea, modificar dicho estado. Por ejemplo, si se trata de un botón, se puede saber si está pulsado con la función *fl_get_button (fd_nombredelForm -> botón)* , los valores que devuelve esta función son *PUSHED* si el botón está pulsado y *RELEASED* si no lo está. Para modificar el estado del botón se utiliza la función *fl_set_button (fd_nombredelForm -> boton, ESTADO)*, siendo el valor de *ESTADO* igual a *PUSHED* o *RELEASED* según lo que se desee.

Para un posicionador también es posible conocer el valor xy del punto donde sus trazas se cruzan. El valor x se obtiene con la función *fl_get_positioner_xvalue (fd_nombredelForm -> posicionador)* y el valor y se obtiene con *fl_get_positioner_yvalue (fd_nombredelForm -> posicionador)*. Para situar el posicionador en un punto se utiliza la función *fl_set_positioner_xvalue (fd_nombredelForm -> posicionador, valor_x)* para indicarle la componente x de la posición, y con la función *fl_set_positioner_yvalue (fd_nombredelForm -> posicionador, valor_y)* se le indica la componente y.

Para conocer el valor actual del elemento *dial* se utiliza la función *fl_get_slider_value (fd_nombredelForm -> slider)*. También es posible determinar su valor desde el programa con la función *fl_set_slider_value (fd_nombredelForm -> slider, valor)*.

Sobre el elemento *canvas* se pueden utilizar funciones que corresponden a la librería *Xlib*. Esta librería está un nivel por debajo de XForms. Para dibujar una línea se utiliza la función *XDrawLine*. En la llamada a esta función hay que especificar el *canvas* donde se desea pintar y las posiciones xy dentro de dicho *canvas* de los puntos de principio y final del segmento. Para dibujar puntos se utiliza la función *XDrawPoints*. A esta función también hay que indicarle el *canvas* donde se desea dibujar los puntos, así como

las posiciones de dichos puntos. En el elemento *canvas* también es posible representar una imagen. Para ello se utiliza la función *XPutImage*, indicándole la imagen que se desea visualizar, el tamaño de la misma y la posición que va a ocupar dentro del *canvas*.

También es posible hacer un repintado de un elemento de un form con la función *fl_redraw_object (fd_nombredelForm -> elemento)*

La interfaz gráfica se inserta en el modelo de programación multitarea descrito como dos tareas más: por un lado muestrea si el usuario ha pulsado algún botón o *slider* y por otro periódicamente refresca la imagen que se le está mostrando. Cuanto más corto sea el periodo de muestreo de los botones antes se entera el programa de la interacción demandada por el usuario y el sistema será más reactivo.

Si la tarea de refresco se llama muy frecuentemente la visualización será muy vivaz, pero puede consumir mucho tiempo y ralentizar al resto de las tareas, lo cual no es deseable. Se ha decidido hacer un refresco cada segundo y resulta ser un buen compromiso entre la viveza de presentación requerida y el ahorro de recursos computacionales que se necesita.

Capítulo 5

El cliente en el PC: Ebase

El sistema teleoperador tiene una arquitectura cliente-servidor con sendos programas, los cuales se comunican entre sí. En el Eyebot se ejecuta el programa servidor Ebase que hemos discutido en el capítulo 3, mientras que en el PC se ejecuta el programa cliente Emovil.

El programa cliente se conecta a través del puerto serie con el servidor para solicitarle información referente a los distintos sensores del robot y para enviarle comandos de movimiento. El usuario desde la interfaz gráfica puede solicitar el valor de los sensores y el programa cliente los representa gráficamente por pantalla. En esta interfaz gráfica también se muestran las medidas tomadas por los infrarrojos, las traslaciones y girosE del robot sobre el plano y las imágenes, tanto en color como en blanco y negro. Además de la visualización de estos elementos la interfaz también ofrece la posibilidad de teleoperar el movimiento del robot de manera que se puede hacerle girar y/o avanzar, y mover la cámara y el pateador a un ángulo deseado.

Todas estas posibilidades de visualización y de teleoperación necesitan de una interfaz intuitiva, capaz de ser utilizada por un usuario no experto. En este caso se ha optado por dotar al sistema de una interfaz gráfica, ya que así se facilita la visualización y la interacción global de todos los elementos. Por ejemplo los valores de los sensores infrarrojos se visualizan como rayos verdes, de longitud proporcional a la medida.

En este capítulo se hace una descripción de los elementos que aparecen en la interfaz gráfica del programa cliente y se aborda la explicación del funcionamiento del programa. Así mismo también se hace un desarrollo del tratamiento de las imágenes, que como ya se comentó en el capítulo 3 tiene ciertas peculiaridades. La visualización de estas imágenes se hace a través de la librería XForms presentada en el capítulo 4, por lo que se hace una explicación del modo en el que se trabaja con ellas. Adicionalmente se

incluye una descripción de los sistemas de coordenadas utilizados para la visualización del Eyebot, su movimiento y las distancias medidas por sus infrarrojos, así como el cambio de un sistema de coordenadas a otro.

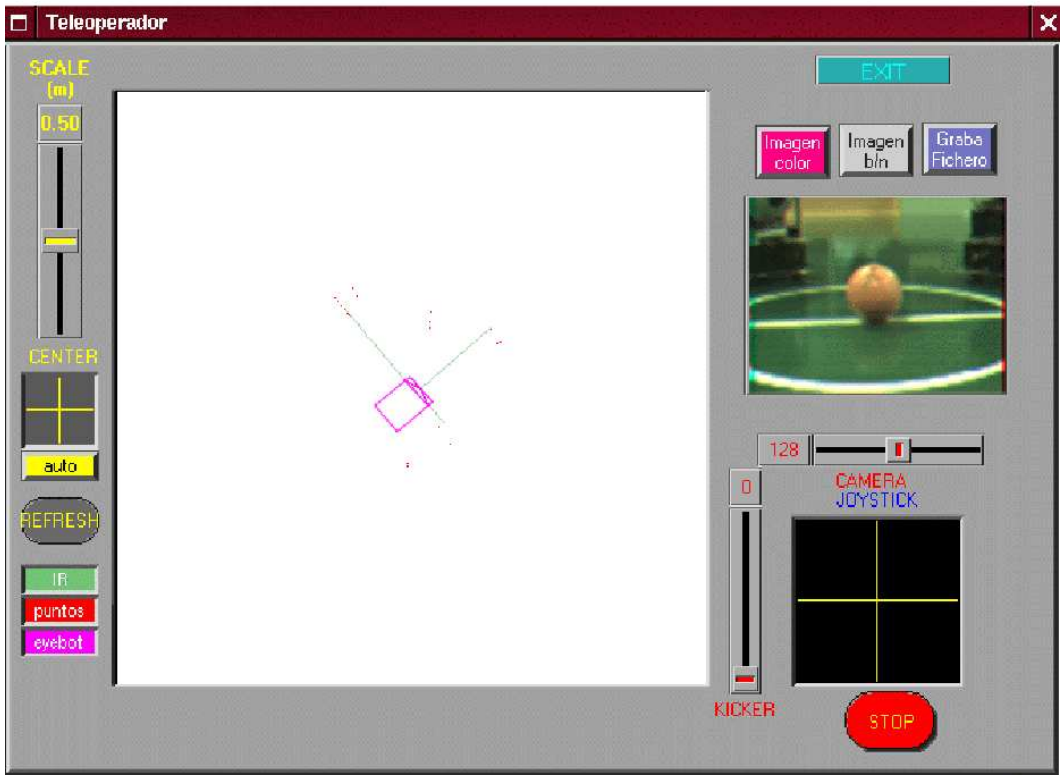


Figura 5.1: Teleoperador

5.1 Elementos de la interfaz gráfica

La interfaz gráfica requerida precisa de elementos que permitan la visualización del robot y su entorno. El robot viene representado en una ventana, en la cual junto a la visualización de su movimiento se pueden seleccionar la representación de otros datos, como los valores de sus infrarrojos.

Las imágenes se pueden solicitar al Eyebot en color o en blanco y negro, siendo necesaria un área de la interfaz para su representación.

La interacción del usuario con los actuadores se lleva a cabo a través de un posicionador para gobernar la traslación y el giro de los motores y de dos *diales*, encargados del anclaje al ángulo indicado de lo servomotores de la cámara y del pateador.

5.1.1 Ventana principal de visualización

Sobre un elemento *canvas* de la librería XForms se ha desarrollado una ventana bidimensional de color blanco que representa el suelo por donde el robot se desplaza. Dentro de esta ventana se muestra el robot y todos los movimientos que realiza, tanto giros como traslaciones, que vienen determinados por las variaciones en sus sensores de posición (encoders).

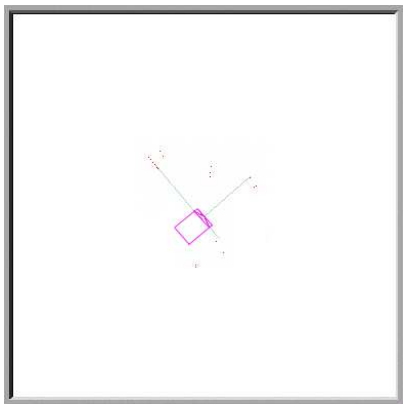


Figura 5.2: Teleoperador: Ventana de visualización 2D

Elementos visualizables.

En la ventana de visualización se observa un mundo bidimensional (5.2) en el cual el robot aparece como un rectángulo magenta con un triángulo inscrito que representa en sus vértices la posición de los sensores infrarrojos, y que permite además apreciar la orientación del Eyebot, diferenciando entre su parte trasera (sin sensor) y su parte delantera (con sensor). Con un botón es posible eliminar la visualización del robot de la ventana. Por defecto éste botón aparece activado, de manera que inicialmente se visualice el robot.

Los obstáculos cercanos al Eyebot están determinados por las medidas de los infrarrojos, las cuales se representan con líneas de color verde que unen el sensor con el punto donde se ha detectado un objeto.

La visualización de los puntos donde se ha detectado un obstáculo se realiza a con puntos rojos. Cada vez que llegan nuevas medidas de los sensores de infrarrojos, estas se almacenan en un buffer circular. Este buffer se va rellenando, y una vez que se completa vuelve a insertar nuevos puntos desde la primera posición. De

Figura 5.3: Ventana de visualización

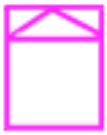


Figura 5.4: Representación del Eyebot sobre la ventana principal

este modo los puntos nuevos van desplazando a los antiguos, pero se conserva una memoria de puntos del pasado reciente.

Las visualizaciones tanto de las medidas tomadas por los infrarrojos como de éste buffer de puntos son opcionales pudiendo activarse o desactivarse pulsando un botón. Por defecto estos botones aparecen desactivados.

Figura 5.5: Teleoperador: Botones de visualización

Configuración de la ventana de visualización 2D.

En ocasiones es necesario hacer modificaciones en el área que abarca la ventana visualización. A ésta ventana se le puede ampliar la escala para abarcar más o menos espacio real, o se le puede hacer un desplazamiento de su centro.

El cambio de escala se realiza a través del elemento *dial* de XForms visto en el capítulo anterior.

Figura 5.6: Teleoperador: Escala

El cambio del centro de la ventana de visualización se puede realizar de manera manual o de modo automático a voluntad del usuario del teleoperador.

Para hacerlo manualmente se ha de actuar sobre un posicionador (elemento de XForms visto en el capítulo 4). Pulsando en los distintos cuadrantes del mismo el centro se desplaza hacia donde se haya pulsado. Estos desplazamientos son siempre traslaciones, no se contempla la posibilidad de realizar un giro en la posición de la ventana 2D.

Se ha desarrollado un cambio automático que cuando el Eyebot se encuentra a menos de un 25% de los límites de la ventana de visualización, ésta se reposiciona para que esté centrada sobre el robot. Esto ocurre así cuando el botón *autotracking* está pulsado.

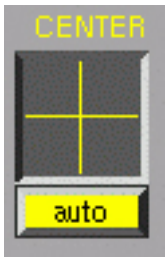


Figura 5.7: Teleoperador: Tracking y autotracking

El programa implementa un repintado periódico de la ventana de visualización y sus elementos para reflejar posibles cambios en la posición u orientación del robot, o reflejar los cambios en el valor de sus sensores infrarrojos. Cuando una ventana se superpone a la interfaz gráfica del teleoperador la visualización de la

misma queda afectada hasta que se produce dicho repintado, por ello es posible forzar un repintado de manera manual a través del botón de refresco.

Figura 5.8: Teleoperador: Botón de refresco

5.1.2 Imágenes

Es posible visualizar las imágenes que está viendo el Eyebot en color o en blanco y negro. La imagen se representa sobre un "*canvas*"(elemento de XForms visto en el capítulo 4) de tamaño 166x126 píxels que ocupa el área superior derecha de la interfaz. La imagen que viene del Eyebot es de 82x62 píxels y resulta demasiado pequeña para una correcta visualización. Aquí se amplía la imagen para representarla con una ampliación de 2x2 para hacerla de tamaño 164x124 píxels. Los píxels sobrantes son utilizados para definir los márgenes en el *canvas*. La captura de las imágenes se habilita a través de dos botones, uno para imágenes en color y otro para imágenes en blanco y negro. Sólo puede estar pulsado uno, ya que sólo es posible visualizar uno de los dos modos a la vez. Para desactivar la visualización se han de desactivar ambos. Al hacerlo, sobre el *canvas* queda representada la última imagen adquirida.

Para facilitar un posterior uso y depuración de las imágenes transferidas pulsando un botón se graba en un fichero la imagen actual. Esta imagen se almacena con el nombre cineN donde N es el número de imagen. El formato del fichero variará en función del tipo de imágenes del que se trate, siendo para imágenes en blanco y negro el formato *pgm* (mapa portable de grises) y para imágenes en color el formato *ppm* (mapa portable de píxels).

5.1.3 Control del movimiento

La interfaz gráfica permite mover el robot a voluntad, hacerle girar, avanzar o ambas cosas a la vez. Para ello se utilizan un posicionador a modo de *joystick* y un botón de *STOP*, ambos son elementos de XForms vistos en el capítulo anterior.

El *joystick* permite graduar el movimiento del robot, de manera que el eje horizontal represente el giro y el eje vertical la traslación, pudiendo hacerse combinaciones entre ambos.

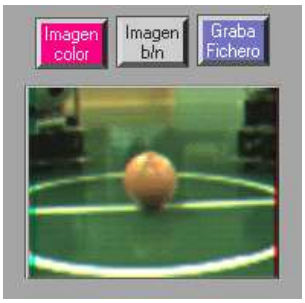


Figura 5.9: Teleoperador: Visualización de imágenes

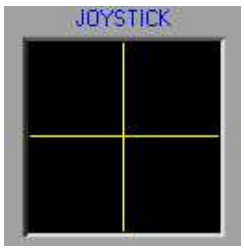


Figura 5.10: Teleoperador: Joystick

La traslación puede hacerse tanto hacia delante como hacia atrás y se ha implementado con un control en velocidad, de manera que la traza que intersecciona con el eje vertical indica la velocidad constante a la que se va a producir el desplazamiento del robot. El centro indica la mínima velocidad y los extremos del eje vertical la máxima. Los giros pueden hacia la izquierda y hacia la derecha y se producen todos a la misma velocidad, ya que sobre ellos se realiza un control en posición.

Cuando se desea detener el robot hay que pulsar el botón *STOP*.

El control de la posición de los servomotores se realiza a través de dos elementos *diales* . Con ellos se fija el ángulo al que se desea anclarlos.

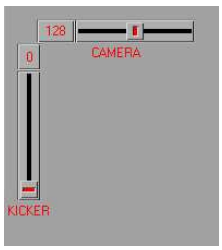


Figura 5.11: Teleoperador: Control de posición de los servomotores

5.1.4 Salida

Al pulsar el botón (elemento de XForms visto en el capítulo anterior) *EXIT* el programa cliente termina. La ventana de la interfaz gráfica se cierra y se le envían al robot los comandos necesarios para ejecutar las desconexiones a las suscripciones y finalizar las comunicaciones con él.

5.2 Implementación

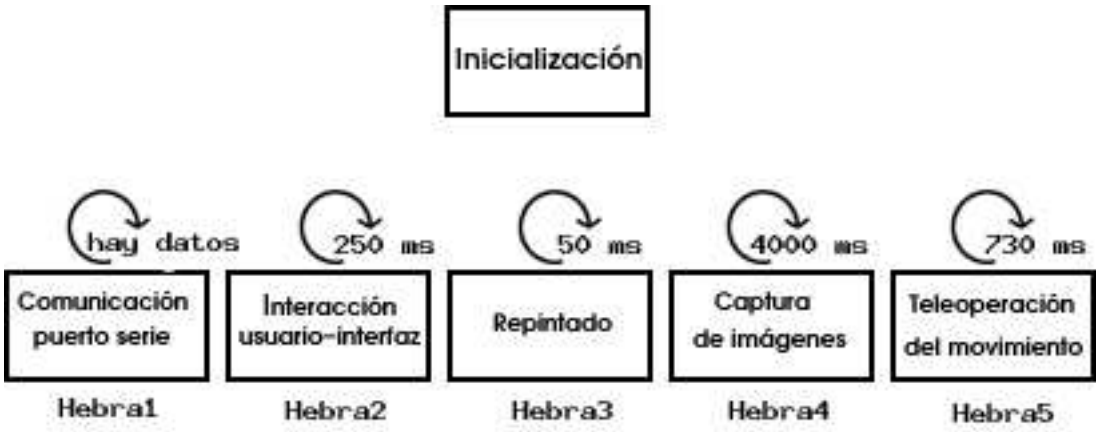


Figura 5.12: Tareas en el cliente

El programa cliente se articula como un conjunto de hebras que empiezan a ejecutarse después de un bloque de inicialización. En este bloque de inicialización es donde el programa:

- Inicializa el puerto serie y establece una velocidad de 115200 baudios para las comunicaciones por el puerto serie (ha de coincidir con la velocidad a la que el Eyebot transmite).

- Prepara la interfaz gráfica, para ello crea el *form* donde se visualizarán los elementos de la interfaz gráfica, e inicializa los elementos de dicho *form* ajustando los valores predeterminados de los mismos, como el color del fondo de la ventana principal de visualización, ajusta la escala, activa el botón de visualización del Eyebot. Además inicializa la posición del Eyebot y de sus elementos dentro de la ventana 2D.

- Inicializa el planificador Rai-Scheduler y los módulos de las distintas tareas que se habrán de ejecutar.

Después de hacer las inicializaciones comienza la comunicación con el Eyebot, para ello ha de ser reconocido por éste, para lo cual se envía un mensaje de saludo (*HOLA\n*). Los movimientos de la representación del Eyebot sobre el *canvas* han de ser fieles a la realidad en todo momento, por lo que tras el mensaje de saludo se envía un mensaje de suscripción a los encoders para controlar el movimiento del robot y representarlo.

Tras estas inicializaciones lo siguiente que hace el cliente es inicializar cinco hebras en el planificador Rai-Scheduler. Una hebra activada por el evento de que haya datos en el puerto serie se encargará de la lectura de dichos datos, componer los mensajes que vayan llegando, analizarlos y actuar en consecuencia.

Otra segunda hebra, ésta activada periódicamente cada 250 milisegundos, se encarga de comprobar si se ha producido una interacción del usuario con la interfaz gráfica.

Una tercera hebra se encarga de hacer un repintado periódico cada 50 milisegundos de los elementos de la ventana de visualización.

Otra hebra periódica cada 4 segundos solicita una imagen en caso de que sea necesario, es decir, si el usuario tiene pulsado uno de los dos botones para visualizar imágenes.

Y por último otra hebra periódica cada 730 milisegundos se encarga de enviar los comandos de movimiento para el Eyebot en caso de que sea necesario.

Después de lanzar el planificador es éste quien toma el control sobre el flujo del programa, que sólo retornará al código principal cuando se finalice dicho planificador, tras lo cual se enviará un mensaje de ADIOS para desconectar con el Eyebot. Hecho esto el programa finalizará.

5.2.1 Hebra para las comunicaciones por el puerto serie

Dentro del planificador de tareas hay una hebra que se activa cuando se produce el evento "hay dato en el puerto serie". Esta función se encarga de la recepción de datos por el puerto serie, realizando lecturas exhaustivas del mismo.

Los bytes recibidos se vuelcan a un buffer donde se analizan buscando mensajes completos o imágenes enteras. Normalmente esta hebra funciona a impulsos de comunicaciones, de manera que son necesarias varias llamadas para recibir todos los bytes de una imagen.

El análisis que se hace de los datos recibidos es distinto si se están recibiendo mensajes o se está recibiendo una imagen. El puerto señaliza todos los mensajes provenientes del Eyebot. Hay dos tipos de mensajes, los normales y las imágenes. Como ya se describió en el capítulo 3 el protocolo de comunicaciones establecido indica que los mensajes tienen una estructura fija del tipo <CABECERA> <CUERPO>\n, mientas que las imágenes pueden incluir cualquier carácter, debiendo ir precedidas de un mensaje. El tipo de transmisión que se esté llevando a cabo (mensajes o imágenes) requerirá un tratamiento distinto de los datos leídos del puerto serie, siendo por tanto necesarios dos modos de lectura.

Un flag señaliza el modo de recepción en el que se encuentra el programa. El cambio del modo de lectura de mensajes a lectura de imágenes se realiza cuando en el análisis de los mensajes se detecta un mensaje de aviso de envío de imágenes. A partir de ese momento se entra en el modo de lectura de imágenes. En las posteriores lecturas llegan bytes que pertenecen a la imagen. El programa contabiliza los píxels de la imagen que le llegan y cuando se completa la imagen se cambia el flag y se vuelve al modo de lectura de mensajes.

Modo de lectura de mensajes Cuando se entra en este modo se realiza una lectura exhaustiva de los datos que haya en el puerto serie. En esta lectura todos los datos que hubiera en él se guardan en un buffer para ser analizados. En ocasiones ocurre que de una vez se ha leído más de un mensaje completo , y que otras veces no se lea ningún dato. Por ello aunque no haya datos en el puerto serie es necesario analizar el buffer de almacenamiento por si hubiera datos de lecturas anteriores.

Teniendo en cuenta el protocolo de comunicaciones descrito en el capítulo 3, el análisis consiste en recorrer el buffer buscando un fin de línea (carácter "\n") porque eso significa que se ha recibido un mensaje entero que es necesario procesar.

Mientras no se completa un mensaje los bytes leídos se almacenan en el buffer de recepción, en espera de completar el mensaje en las siguientes lecturas. Una vez que se ha obtenido un mensaje completo es necesario darle servicio. En primer lugar se extrae una cabecera que permita determinar el tipo de mensaje del que se trata y así poder actuar en consecuencia.

1. Si se trata de una cabecera ADIOS se procede a finalizar el planificador, lo que origina el retorno a la función principal y en consecuencia la finalización del programa cliente. El programa servidor fuerza de este modo la terminación del

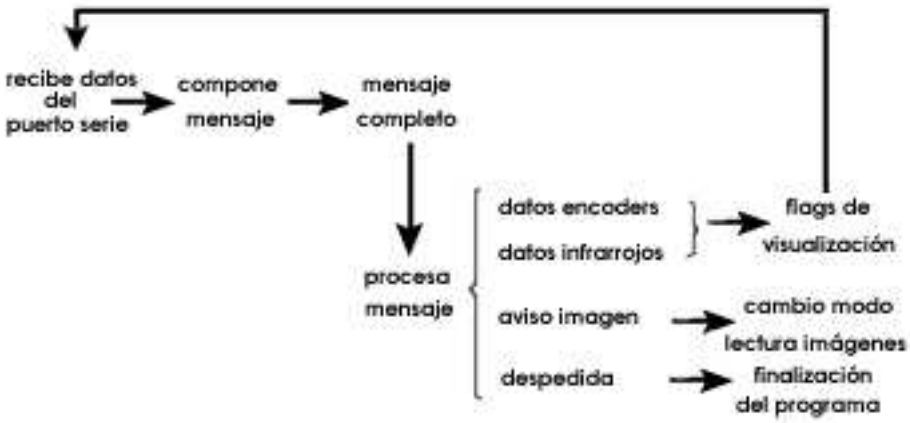


Figura 5.13: Hebra para las comunicaciones: Modo de lectura normal

sistema y el cierre de las comunicaciones.

2. Si la cabecera es TOMAENCODER se analiza el resto del mensaje para obtener los valores de los encoders izquierdo y derecho. Estos valores se comparan con los valores anteriores para ver si se ha producido algún desplazamiento del robot. Si éste no se ha movido no se hace nada, pero si lo ha hecho es necesario calcular la nueva posición y orientación gráficas del Eyebot sobre la ventana de visualización 2D. Para estos cambios de coordenadas se utiliza la metodología explicada en la sección "Sistemas de coordenadas" este mismo capítulo.

3. Si la cabecera es TOMAINFRARROJO se extrae del mensaje los valores de los 3 infrarrojos. Estos valores son cambiados de sistema de referencia, ya que para poder representarlos es necesario referenciarlos al sistema de coordenadas de la ventana de visualización. El cambio de coordenadas se explica en el apartado "Coordenadas" de este mismo capítulo.

4. Si es una cabecera del tipo TOMAIMAGENGRISASCII, TOMAIMAGENGRISBIN, TOMAIMAGENCOLORASCII o TOMAIMAGENCOLORBIN se procede a activar los flags que indican que se va a recibir una imagen. En esos momentos todo lo que haya recibido y se reciba después de esta cabecera es parte de una imagen, por lo que es necesario realizar un cambio de modo de lectura. El resto de datos almacenados en el buffer después de éste mensaje pasan a formar parte del buffer de la imagen.

Si el mensaje recibido es un envío de los valores de los infrarrojos o de los encoders (un mensaje normal) el modo de lectura no varía. Los mensajes a los que ya se ha dado servicio son desechados. En una misma lectura se puede recibir más de un mensaje, por lo que es necesario analizar el resto de datos que pudiera haber en el buffer de entrada. El procedimiento a seguir para el análisis de estos datos es idéntico al descrito hasta ahora.

Modo de lectura imagen Si el mensaje que se ha recibido es un anuncio de que se va a enviar una imagen entonces se cambia al modo de recepción de imágenes. Esto es así para asegurar la trasparencia de datos, de manera que una imagen pueda incluir tantos "\n" como necesite sin que esto afecte al funcionamiento de las comunicaciones. Los bytes que se hubieran recibido después del aviso de imagen forman parte de la propia imagen y son copiados en el buffer de imágenes.

En el mensaje de aviso de envío de imagen se mandan los datos necesarios para poder determinar el número de datos que se han de esperar como parte de la imagen. Las imágenes que pueden ser enviadas a través del puerto serie son de dos tipos, según la naturaleza de los datos que la componen, y pueden ser ASCII o binarias. Dependiendo de si se está transmitiendo una imagen binaria o ASCII el control de los datos transmitidos es diferente.

- Para imágenes binarias se cuenta el número de píxels transmitidos como el número de datos recibidos, ya que para cada píxel se envía su valor en un sólo carácter para imágenes en blanco y negro, o en tres caracteres (componentes RGB) para imágenes en color. Por tanto se ha de esperar recibir un total de bytes igual a

$$bytes = filas * columnas * Bpp$$

  Siendo filas y columnas las dimensiones de la imagen y Bpp el valor que indica si se está recibiendo una imagen a color (Bpp = 3) o en blanco y negro (Bpp = 1).

- Para imágenes transmitidas en ASCII no se puede establecer a priori el número de bytes que se transmiten por imagen, por tanto se envía una línea por píxel para imágenes en escala de grises, o una línea por componente de píxel para imágenes en color. Esto es, si el valor de un píxel es 156, primero se enviará el 1, después el 5 y por último el 6, seguido de un "\n". Por tanto lo que se han de analizar los datos recibidos contabilizando los "\n" que delimitan los valores

de los píxels (o de sus componentes), de manera que se espere recibir un total de caracteres "\n" igual a

$$lineas = filas * columnas * Bpp$$

Este modo de lectura está pensado para transmitir un determinado número de bytes. Cuando entra en este modo lo que se hace es comprobar que haya datos en el puerto serie, y en caso afirmativo realiza la lectura almacenando los datos en el buffer de imágenes (no en el buffer de mensajes como se hacía en el otro modo de lectura). Una vez que se ha leído el puerto serie hay que buscar los "\n" (imágenes en ASCII) o contar los bytes enviados para saber el número de píxels que se llevan transmitidos hasta completar la longitud que se ha determinado que ha de tener la imagen.

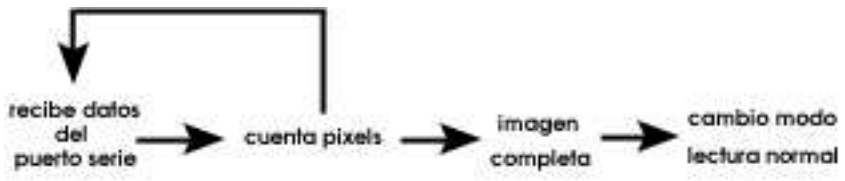


Figura 5.14: Hebra para las comunicaciones: Modo de lectura de imágenes

Una vez completada la imagen se llama a la función *ProcesaImagen* si se trata de una imagen ASCII o a *ProcesaImagenBin* si es binaria. Estas funciones se encargan de la visualización en la interfaz gráfica .

La manera de transmitir imágenes más eficiente es en modo binario. Esta transmisión ocupa un menor ancho de banda, ya que requiere enviar un menor número de bytes para transmitir cada imagen. Aunque en el teleoperador es posible el envío / recepción de ambos tipos de imágenes el modo binario es el que se ha utilizado.

Tras completar la imagen el modo de lectura cambia al modo de lectura de mensajes. En una misma lectura pueden llegar el final de una imagen y otros mensajes, por lo que los datos que se hayan leído después del último byte que corresponde a la imagen pasan a formar parte del buffer de almacenamiento de mensajes, para ser analizados seguidamente.

Si no se ha completado ninguna imagen los bytes leídos se incorporan al buffer de imágenes y se sale de la función, esperando nuevos datos para completar la imagen. La próxima vez que se ejecute ésta función volverá a entrar en el modo de lectura imagen.

5.2.2 Hebra para el repintado de la interfaz gráfica

El usuario a través de la interfaz gráfica puede ver la posición y orientación que tiene el robot y los valores de sus sensores. Estos datos han de estar permanentemente actualizados, de manera que si por ejemplo el robot gira, en la ventana de visualización 2D se observe esta variación. Es necesario pues repintar periódicamente los elementos que aparecen representados en la interfaz gráfica para que estén actualizados respecto a la realidad. Para ello se ha implementado una hebra periódica. Esta hebra se encarga de pintar los distintos elementos dentro de la ventana principal de visualización de la interfaz gráfica (el robot, su movimiento, las medidas de los infrarrojos y los obstáculos detectados con anterioridad).

Esta hebra no hace un repintado de la imagen, ya que como máximo se recibe una imagen cada 4 segundos. Si se repintara la imagen cada 50 milisegundos representaría un coste computacional innecesario. Las imágenes se actualizan cuando se terminan de recibir, y ya no se vuelven a repintar hasta que no se reciba otra.

El usuario configura a través de la interfaz gráfica que elementos quiere que se le muestren. Esta configuración se almacena en forma de flags, y en función de estos flags se repintarán unos objetos u otros. Esta interacción del usuario es controlada a través de una hebra que se verá más adelante, y que será la encargada de actualizar los flags para que la hebra de repintado actúe convenientemente.

Para que la hebra sea más eficiente se producen dos tipos de repintado. Hay un repintado incremental, válido cuando la inserción de nuevos elementos no altera demasiado la visualización de los anteriores. Esto ocurre con los puntos rojos que indican dónde se ha detectado un objeto. Estos puntos forman parte de un buffer, de manera que en un repintado incremental sólo se actualizan los nuevos puntos. En un repintado incremental no se borra ningún elemento que estuviera dibujado con anterioridad. El otro tipo de repintado es un repintado total, en el que se borran todos los objetos que estuvieran dibujados sobre la ventana y se dibujan los nuevos.

En esta hebra se comprueba si es necesario hacer un repintado incremental cada 50 milisegundos. Cada treinta veces que se ejecute (150 milisegundos) el repintado que se hace es total. Adicionalmente es posible reforzar un repintado total desde la interfaz gráfica a través del botón *REFRESH*.

Tal y como ya se comentó cuando se explicaron los elementos que formaban la interfaz gráfica es posible modificar la posición del centro de la ventana de visualización utilizando el *tracking* y el *autotracking*, así como modificar la escala de la misma a través

del *dial* destinado a tal función. Cada vez que el *tracking* y la escala son modificados por el usuario se fuerza un repintado total para adaptar los elementos visualizados a las nuevas condiciones de la ventana. Además, si se tiene pulsado el botón de autotracking si el robot está a menos de un 25% del borde de la ventana, el centro de ésta se situará sobre el robot, por lo que también se forzará el repintado total automático.

Cuando se realiza un repintado total de los distintos elementos, primero se borran los actuales pintándolos en color blanco. Es decir, si el robot se está moviendo, antes de pintar la nueva representación con el color correspondiente se debe de haber pintado en blanco la representación anterior.

Todos los puntos que se van a representar en la ventana de visualización 2D, ya sea porque pertenezcan al robot, o porque sean la medida de los sensores de infrarrojos, o formen parte de la memoria de puntos, han de ser referenciados al sistema de coordenadas de la ventana de visualización. Para hacer esta conversión, primero hay que referenciar todos los puntos al sistema de coordenadas del robot, y después de éste al sistema de referencia del mundo. Una vez que se tienen los puntos calculados respecto del mundo hay que referenciarlos respecto de la ventana de visualización, que es donde finalmente se van a representar. Este cambio de sistemas de referencia se explica en la sección "Sistemas de coordenadas" este mismo capítulo.

Dentro de la ventana de visualización los posibles elementos a modificar son:

- Robot: Si hay que representar o borrar el robot en el *canvas* lo primero que se hace es borrar la anterior representación del Eyebot y si el flag que indica que hay que representar el robot está activado se pinta la nueva representación. Para hacerlo se debe rellenar una estructura con los vértices y puntos significativos del robot. Cada punto representativo debe de ser cambiado de sistema de referencia, de manera que se obtenga el punto referenciado respecto del sistema de referencia de la ventana de visualización. Una vez que se tienen estos puntos se unen dibujando las líneas correspondientes, con ello se consigue dibujar el perfil 2D del robot.

- Infrarrojos: Si hay que pintar o borrar la representación de los infrarrojos primero se borran las líneas verdes que indican las distancias a los obstáculos detectados para la anterior representación, y si el botón *infrarrojos* está pulsado se calculan los puntos correspondientes a cada sensor respecto del sistema solidario a la ventana de visualización. Hecho esto se pintan de verde las líneas que unen cada

sensor con el obstáculo que se haya detectado.

- Puntos: La representación de estos datos sobre el *canvas* se realiza únicamente si el flag que permite esta visualización está activado (está pulsado el botón correspondiente). En este caso se puede hacer un pintado total de todos los puntos del buffer o hacer un pintado incremental. El pintado incremental se realiza cuando haya valores nuevos en el buffer, independientemente de si se tiene que hacer el refresco o no. El pintado total del buffer se hace durante el repintado total de todos los elementos, siempre y cuando esté activado el flag para su visualización.

5.2.3 Hebra para la interacción entre el usuario y la interfaz gráfica

Cuando el programa cliente se está ejecutando es necesario que éste pueda comprobar si el usuario ha producido alguna modificación en los objetos de la interfaz gráfica del sistema teleoperado. Con esta misión se ha incluido una hebra dentro del planificador, de manera que periódicamente, cada 250 milisegundos, comprueba si hay algún tipo de interacción entre el usuario y los objetos gráficos.

La elección de este tiempo como periodo de ejecución viene determinada por el hecho de que el sistema debe de producir una sensación de interactividad continua. Con este tiempo de periodo se consigue que como máximo pasen 250 milisegundos desde que el usuario modifica algún objeto hasta que se produce la respuesta del sistema. Si este tiempo fuera mayor, se perdería la sensación de respuesta automática, y si por el contrario el tiempo fuera menor se sobrecargaría innecesariamente la CPU sin que el usuario apreciara una diferencia notable.

Esta hebra es la encargada de actualizar los flags y variables que gobiernan el comportamiento de la hebra de repintado, la de captura de imágenes y la de teleoperación del movimiento del robot.

Como ya se ha visto en este mismo capítulo, la interfaz gráfica ofrece múltiples opciones de interacción con el sistema, botones de visualización, cambio de escala, petición de imágenes, teleoperación del movimiento, etc. Cada una de estas acciones está controlada por un elemento gráfico que puede ser modificado por el usuario por medio del ratón. La biblioteca XForms se encarga de modificar el estado del elemento cuando el usuario pulsa con el ratón sobre él. Además devuelve al programa qué elemento gráfico ha sido accedido por el usuario. El programa puede comprobar el

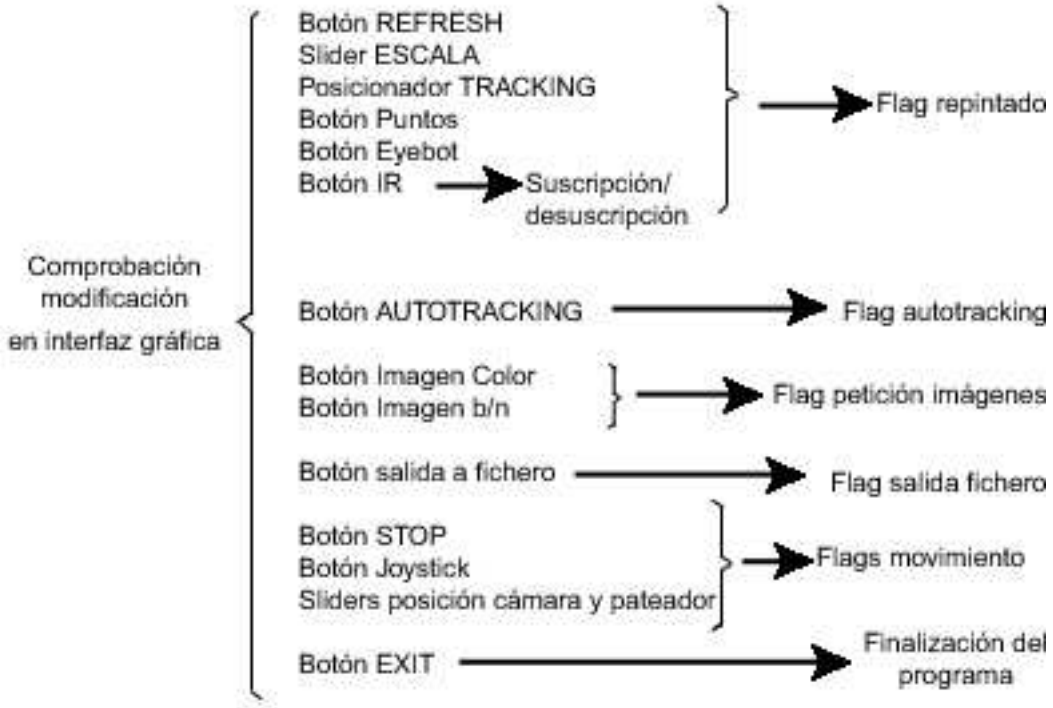


Figura 5.15: Hebra de comprobación de la interacción usuario-interfaz gráfica

estado del elemento correspondiente y actuar en consecuencia. En función del elemento sobre el que se actúe las acciones de respuesta varían.

1. Si se ha pulsado el botón *EXIT* se procede a detener el planificador, por lo que el programa finaliza.

2. Si se ha modificado el valor del objeto *dial* se captura el nuevo valor y se calcula la nueva escala, forzándose un repintado del *canvas* principal y sus elementos que utiliza este nuevo valor.

3. Si se ha pulsado el botón de refresco se fuerza un repintado del *canvas* y de los elementos que en él aparecen.

4. Si se ha modificado el posicionador de la ventana de visualización, ésta se desplaza hacia el lugar indicado.

5. Si se trata de una modificación del botón de *autotracking* se comprueba si está pulsado o no, activando o desactivando convenientemente el flag que lo indica para permitir el seguimiento automático del Eyebot.

6. Si se ha modificado el botón *Infrarrojos*, se comprueba si está pulsado o no. Si está pulsado se envía una suscripción a los infrarrojos y si no lo está se envía una des-suscripción a los mismos.

7. Si el botón *Puntos* se ha modificado se comprueba si está pulsado o no, y en consecuencia se activa o se desactiva un flag que lo indica.

8. Si el botón *Eyebot* ha sido modificado se comprueba si está pulsado o no, y en consecuencia se activa o desactiva un flag que lo indica.

9. Cuando se pulsa el botón *STOP* se envían al robot los comandos necesarios para que pare los motores. Además él *joystick* se coloca en la posición inicial en caso de estar fuera de ella.

10. Si se ha modificado la posición de las trazas del *joystick* se activa un flag para que se envíen los comandos de movimiento en la hebra correspondiente.

11. De los dos botones que solicitan la visualización de imágenes sólo puede estar activado uno a la vez. Si se pica el botón que solicita que se muestren imágenes en color se comprueba su estado. En el caso de que esté activado, si el otro botón estaba activo se desactiva, y se pasa a solicitar imágenes en color. Si se comprueba que el botón modificado está desactivado se dejan de pedir las imágenes correspondientes.

12. En caso de que el botón que solicita que se muestren imágenes en escala de grises sea modificado se comprueba si está activado. Si lo está se desactiva el botón para mostrar imágenes en color, y se activa el flag para solicitar imágenes en escala de grises. Si el botón modificado está desactivado se dejan de pedir las imágenes correspondientes.

13. Si el botón de salida a fichero ha sido modificado se comprueba su estado. Si está pulsado se habilita el flag que lo indica, y la siguiente imagen se graba en un fichero después de ser visualizada en la interfaz gráfica.

5.2.4 Hebra para la captura de imágenes

Esta hebra es la encargada de realizar la solicitud de imágenes al Eyebot. Se trata de una tarea periódica que se ejecuta cada 4000 milisegundos. El motivo de ejecutarse a ese ritmo es doble:

- Si se solicitaran imágenes con más frecuencia de la que es capaz de digerir el canal, se saturaría el buffer de comunicaciones y se produciría un bloqueo del sistema de comunicaciones.

- El volumen de datos de una imagen es muy superior al de un mensaje. Si cada vez que se acaba de recibir una imagen inmediatamente solicitamos otra, el sistema de comunicaciones estará la mayor parte del tiempo transmitiendo imágenes, por tanto los mensajes con los valores de los infrarrojos y de los encoders no llegan a un ritmo constante, sino amontonados entre imagen e imagen. Si esto ocurriera así no se conseguiría la sensación de realismo y continuidad pretendida, y en la interfaz el movimiento del robot se observaría a saltos.

El control sobre las imágenes que se solicitan se ha de llevar a cabo de manera que entre las imágenes puedan llegar los datos necesarios del resto de sensores. El objetivo es que se mantengan unos niveles de correspondencia aceptables entre lo que se está representando en la ventana de visualización y lo que realmente está pasando en el robot. Esta hebra materializa el control de flujo en la transmisión de imágenes y además deja hueco en el canal para mensajes de otro tipo.

Con este motivo, si el flag que indica que uno de los dos botones para solicitar la captura de imágenes está activado se comprueba si realmente se está recibiendo alguna imagen antes de solicitar otra. En caso de que sea así no se enviará una nueva petición de imagen, debiendo esperar hasta la siguiente ejecución de la hebra para poder realizar otra vez esta comprobación.

Si no se está recibiendo ninguna otra imagen, la hebra envía una petición de imagen, comprobando el flag del tipo de captura que se solicita (imágenes en blanco y negro o en color), enviando el mensaje *DAMEIMAGENGRISBIN*$\backslash n$ para escala de grises o *DAMEIMAGENCOLORBIN*$\backslash n$ para imágenes en color. Sólo se envían peticiones de imágenes binarias porque como ya se ha visto son más pequeñas que las imágenes ASCII, y por tanto se tarda menos en transmitirlas.

Si los flags indican que no hay ningún botón pulsado en la interfaz gráfica que solicite la captura de imágenes no se realizará ninguna de las acciones anteriores.

Ejecutando esta hebra cada 4000 milisegundos se deja suficiente hueco para que los mensajes de infrarrojos y encoders lleguen con continuidad.

5.2.5 Teleoperación del movimiento

Esta hebra materializa la teleoperación remota desde un *joystick* gráfico situado en la interfaz.

Cada 730 milisegundos, esta hebra se encarga de comprobar si ha habido alguna modificación en el *joystick*. En caso afirmativo envía al robot el mensaje con los valores para el giro y la velocidad de traslación calculados a partir de la nueva posición de las trazas del *joystick*.

En el capítulo 2,referente al hardware y software del Eyebot, se comentaron las limitaciones del robot en cuanto a las comunicaciones a través del puerto serie. Como ya se comentó las interrupciones originadas por la cámara bloquean la recepción por el puerto serie, para lo cual se optó por inicializar la cámara únicamente cuando se necesite hacer una captura de una imagen, liberando el recurso después. A pesar de esta medida en momentos muy puntuales se produce un error en la recepción por el puerto serie. Esto se debe a los tiempos empleados por el robot para la inicialización y la liberación del recurso y que por tanto no son controlables por el programador.

Es por ello por lo que se hace necesario un control en el flujo de mensajes que se le envían al robot. Con el funcionamiento de la cámara descrito anteriormente el flujo máximo de mensajes que se pueden enviar al Eyebot es de aproximadamente 78 mensajes por minuto. Además de esta hebra, hay otras que envían mensajes, como la solicitud de imágenes o la suscripción o des-suscripción a los sensores infrarrojos. Por tanto se ha establecido un periodo de 730 milisegundos para la ejecución de esta hebra, de manera que el número de mensajes que se le puedan enviar al Eyebot por minuto esté controlado.

Utilizando el *joystick* sobre el movimiento del robot se tiene un control en velocidad de tracción y un control de posición en el giro. Esto en la práctica se materializa en que el eje vertical del posicionador representa la velocidad de tracción, y el eje horizontal el ángulo de giro. Ambos movimientos se pueden realizar de manera independiente o coordinados.

5.3 Sistemas de coordenadas

5.3.1 Sistemas de referencia

La representación del Eyebot y la percepción gráfica de las variables que devuelven sus sensores hacen necesaria la utilización de distintos sistemas de coordenadas. Por

un lado las medidas de los infrarrojos están tomadas respecto del sistema del robot, y a su vez éste se puede desplazar por el sistema mundo, y al hacerlo deberá también desplazarse por el sistema de la ventana de visualización. A su vez la ventana de visualización también se puede desplazar por el sistema mundo.

Tenemos por tanto tres sistemas de referencia que son el del mundo, el gráfico y el solidario al robot.

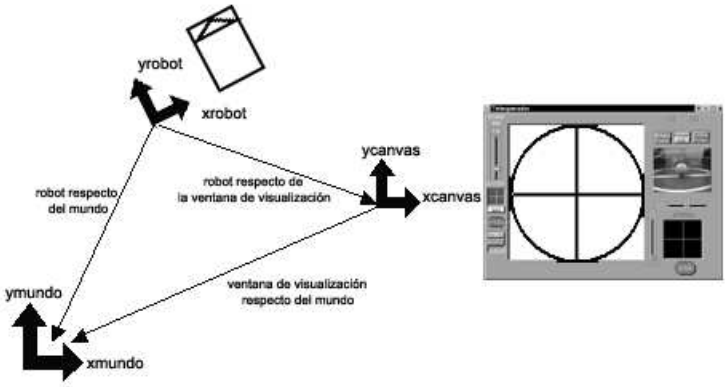


Figura 5.16: Sistemas de referencia

Cuando se quiere representar sobre la ventana de visualización 2D un punto del robot o el punto donde se ha detectado un obstáculo será necesario referenciar este punto sobre el sistema de referencia de dicha ventana. Los pasos a seguir para representar un punto en la ventana 2D son:

- Se ha de conocer la posición del punto respecto del sistema de referencia del robot. Para cada punto del robot habrá un sistema de referencia con el eje x+ normal al robot. De esta manera los valores de *Xp_punto* y de *Yp_punto* serán *0* si se trata de un punto perteneciente al robot. Si se trata de un punto donde se ha detectado un obstáculo por los infrarrojos *Xp_punto* tendrá el valor de la distancia medida, mientras que *Yp_punto* tendrá el valor *0*, ya que el sensor detecta un obstáculo que está frente a él.

$$Xp\_robot = X\_sensor + Xp\_punto\cos\alpha - Yp\_punto\sin\alpha$$
$$Yp\_robot = Y\_sensor + Yp\_punto\cos\alpha + Xp\_punto\sin\alpha$$

  Donde α el ángulo del sensor respecto del sistema de referencia del robot. *X_sensor* e *Y_sensor* es la posición del sensor respecto del sistema de referencia del robot.

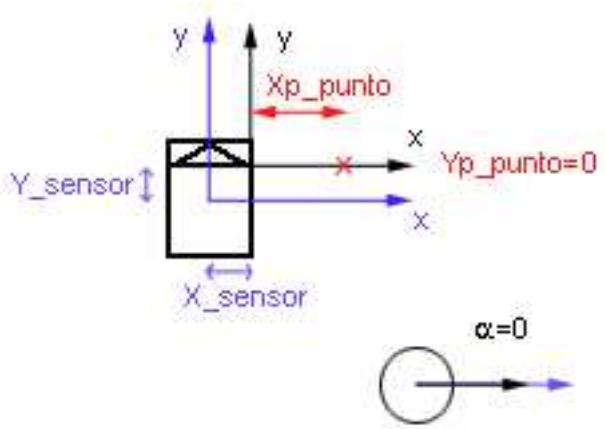


Figura 5.17: Un punto respecto del sistema de referencia del robot

- Una vez conocida la posición del punto respecto del robot es necesario calcular esta posición respecto del sistema del mundo. Es necesario hacer un cambio de sistemas de referencia desde el sistema del robot al sistema del mundo.

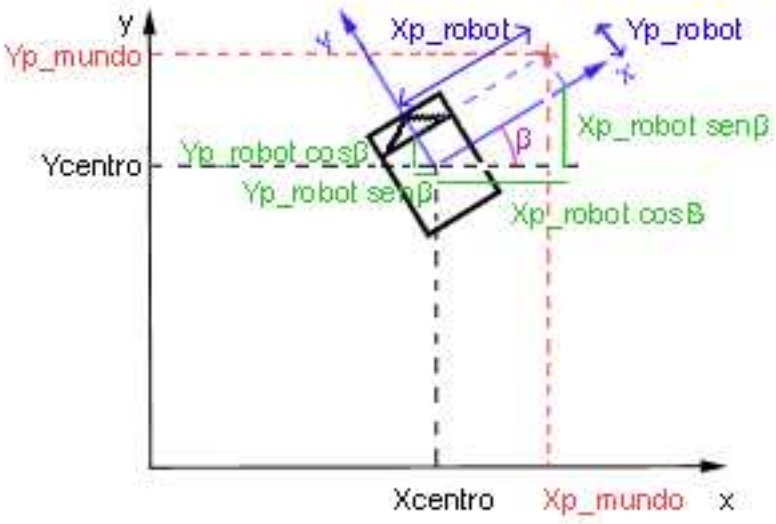


Figura 5.18: Cambio desde el sistema de referencia del robot al del mundo

$$Xp\_mundo = Xp\_robot\cos\beta - Yp\_robot\sin\beta + Xcentro$$
$$Yp\_mundo = Yp\_robot\cos\beta + Xp\_robot\sin\beta + Ycentro$$

Donde β es el ángulo entre el sistema de referencia del mundo y el sistema de

referencia del robot. *Xcentro* e *Ycentro* es la posición del centro del sistema de referencia del robot respecto del sistema mundo.

- El punto ha de ser representado en la ventana de visualización. Para que esto se haga correctamente ha de estar referenciado respecto del sistema de referencia de la ventana 2D. Para referenciar el punto al sistema de la ventana se ha de hacer un cambio de sistemas de referencia desde el sistema del mundo al de la propia ventana 2D.

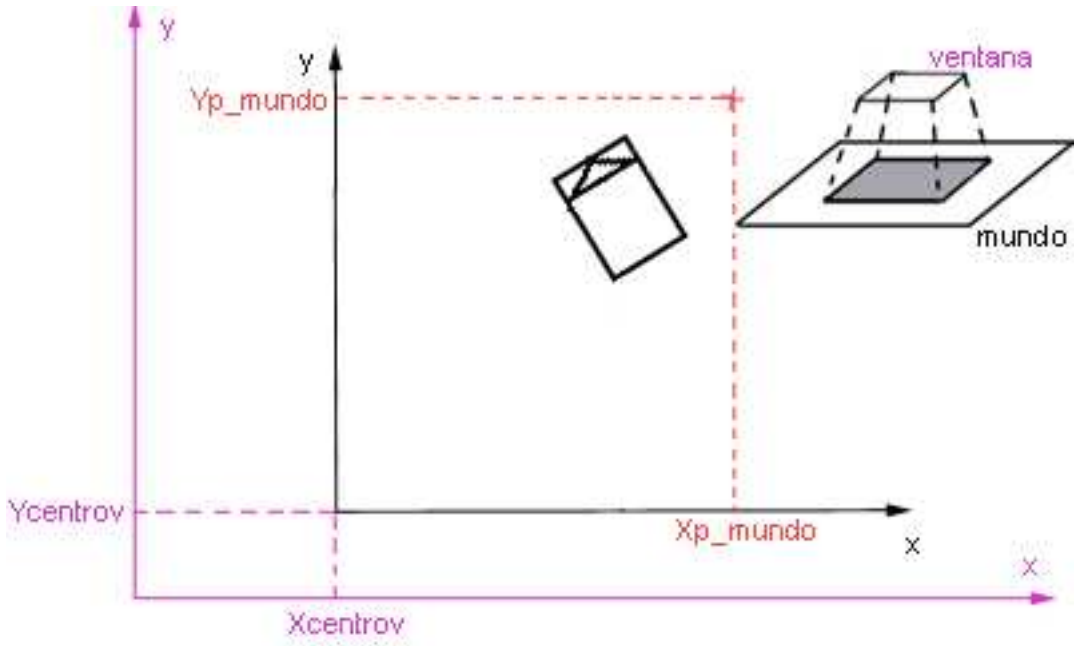


Figura 5.19: Cambio al sistema de referencia de la ventana de visualización 2D

$$Xp\_ventana = (Xp\_mundo \cos\Delta) - Yp\_mundo \sin\Delta + Xcentrov) * escala$$
$$Yp\_ventana = (Xp\_mundo \sin\Delta) + Yp\_mundo \cos\Delta + Ycentrov) * escala$$

Δ es el ángulo entre el sistema de referencia de la ventana de visualización y el sistema de referencia del mundo. *Xcentrov* e *Ycentrov* es la posición del centro del sistema de referencia del mundo respecto del sistema de la ventana 2D.

Hay que tener en cuenta que se debe realizar un ajuste final, ya que en la ventana de visualización la posición $(x, y) = (0, 0)$ corresponde con la esquina superior izquierda, y se desea que se sitúe en el centro, por lo que se deberá desplazar la coordenada x un valor de $\frac{width}{2}$ y la coordenada y un valor de $\frac{height}{2}$, siendo *width*

y height el ancho y el alto de la ventana.

$$X = (int)Xp_ventana + \frac{width}{2}$$
$$Y = (int) - Yp_ventana + \frac{height}{2}$$

La posición y orientación del robot respecto de mundo se almacenan en variables que son actualizadas cada vez que llega información del movimiento del robot a través del valor de sus encoders. También se almacena la posición y orientación de la ventana de visualización respecto del mundo en variables que se actualizan cuando se produce un cambio en la escala o una modificación del centro de la ventana 2D.

En el programa cliente las dos funciones encargadas de realizar los cambios de sistemas de referencia son *ir2xy* y *xy2canvas*. La función *ir2xy* calcula la posición respecto al sistema de referencia del robot tanto de los puntos de éste como de los puntos donde se ha detectado un obstáculo. Una vez que conoce esta posición respecto del sistema del robot, la calcula respecto del sistema de referencia del mundo. La función *xy2canvas* realiza el cambio del sistema de referencia del mundo al de la ventana de visualización.

5.3.2 De los encoders a la interfaz gráfica

La posición y orientación del robot viene dada por los pulsos de los encoders de los motores izquierdo y derecho del robot. Esta información no es directamente la posición x, y, θ que se necesita para representar el robot sobre la ventana de visualización 2D. En apartado se describe cómo obtener la posición y orientación del robot desde el valor de los encoders.

Cuando el robot se mueva la trayectoria que realice deberá ser representada en el *canvas* con un movimiento análogo en la ventana 2D que se corresponda con el real. La indicación de este desplazamiento viene determinada por los pulsos que devuelven los encoders. Conociendo la posición $x0, y0, \theta0$ del robot en un instante determinado, se quiere calcular la posición $x1, y1, \theta1$ en el instante siguiente (muy cercano en el tiempo). Para calcular la nueva posición se conoce los pulsos que ha devuelto cada encoder, por lo que se puede calcular la distancia recorrida por cada rueda. En la tabla HDT se calibra el número de pulsos que devuelve cada encoder tras recorrer un metro, en este caso son 3240 pulsos. Para calcular el desplazamiento de cada rueda se realizará el siguiente cálculo (5.1).

$$\Delta S_x = \frac{k}{3240} \quad \frac{pulsos}{\frac{pulsos}{metro}} \tag{5.1}$$

Siendo k el número de pulsos devuelto por el encoder correspondiente.

Asumiendo una velocidad constante tanto lineal como rotacional, el cambio en la posición y orientación del Eyebot está dado por la relación de la suma y de la diferencia, respectivamente, de la distancia recorrida por cada rueda. Esta asunción es muy acertada para intervalos de tiempo pequeños, como es el tiempo entre lectura y lectura de los encoders.

En el intervalo Δt te se considera que el Eyebot está recorriendo un arco de radio R, que abarca un ángulo $\Delta\alpha$, teniendo en cuenta que la distancia entre ruedas es w, la rueda izquierda recorre un arco ΔS_i de radio $R - \frac{w}{2}$, mientras que la rueda derecha recorre un arco ΔS_d de radio $R + \frac{w}{2}$

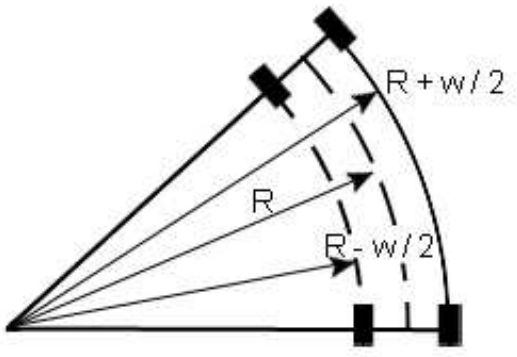


Figura 5.20: Desplazamiento

La longitud del arco ΔS está relacionada con el ángulo α y con el radio R por la fórmula

$$\Delta S = \Delta\alpha R \tag{5.2}$$

.

Por tanto,

$\Delta S_i = \Delta\alpha(R - \frac{w}{2})$ será el desplazamiento de la rueda izquierda.

$\Delta S_d = \Delta\alpha(R + \frac{w}{2})$ será el desplazamiento de la rueda derecha.

Si se resta estas relaciones se obtiene:

$$\Delta S_d - \Delta S_i = \Delta\alpha(R + \frac{w}{2}) - \Delta\alpha(R - \frac{w}{2}) = \Delta\alpha w$$

O lo que es lo mismo, el ángulo α del arco recorrido es

$$\Delta\alpha = \frac{\Delta S_d - \Delta S_i}{w} \tag{5.3}$$

El arco recorrido por el punto medio entre ruedas está dado por la suma de las distancias recorridas por cada rueda.

$$\begin{aligned} \Delta S_d + \Delta S_i &= \Delta\alpha(R + \tfrac{w}{2}) + \Delta\alpha(R - \tfrac{w}{2}) \\ &= 2\Delta\alpha R \\ &= 2\Delta S \end{aligned}$$

Con lo que se obtiene que el desplazamiento del punto medio entre las ruedas es igual a

$$\Delta S = \frac{\Delta S_d + \Delta S_i}{2} \tag{5.4}$$

Aplicando (5.2) el radio del arco recorrido por el punto medio es $\frac{\Delta S}{\Delta\alpha}$

Para poder determinar el centro de giro se necesita saber es si la traslación se ha producido en el sentido de las agujas del reloj o en su contra. Para ello hay que comparar los valores absolutos de los desplazamientos de cada rueda. Si $|\Delta S_i| < |\Delta S_d|$ la rueda derecha se ha movido más y por lo tanto el centro de giro está a la izquierda del robot. Si por el contrario $|\Delta S_i| > |\Delta S_d|$ el centro de giro está a la derecha del robot.

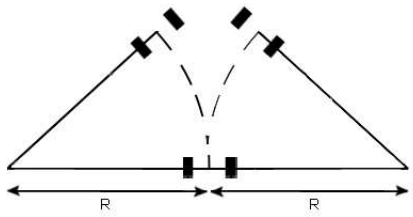


Figura 5.21: Arcos posibles para un mismo radio

El centro de giro se calcula aprovechando por un lado que el centro de giro está en la recta que contiene al centro del robot y a su rueda izquierda y derecha, y por otro lado que la distancia entre el centro de giro y el centro del robot es el radio de giro R.

Si se llama "a" al centro del robot , una recta que uniera este punto con otro genérico tendría las siguientes ecuaciones paramétricas.

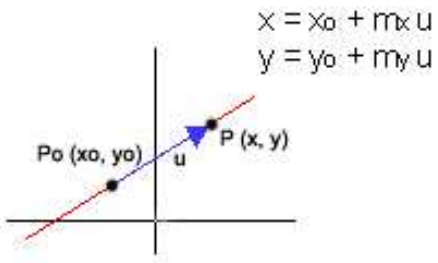


Figura 5.22: Ecuaciones paramétricas de la recta

$$\begin{aligned} y &= y_a + m_y\lambda \\ x &= x_a + m_x\lambda \end{aligned} \tag{5.5}$$

Siendo λ la distancia desde el punto a al punto genérico. Si esta recta pasara por el punto b situado en una rueda entonces:

$$\begin{aligned} y_b - y_a &= m_y\lambda \\ x_b - x_a &= m_x\lambda \end{aligned} \tag{5.6}$$

Como la distancia λ entre a y b es conocida $\frac{w}{2}$, es posible calcular los valores para m_x y m_y que serán constantes.

$$\begin{aligned} (y_b - y_a)\tfrac{2}{w} &= m_y \\ (x_b - x_a)\tfrac{2}{w} &= m_x \end{aligned} \tag{5.7}$$

La distancia entre el centro del robot y el centro de giro es el radio R de giro. Utilizando el teorema de Pitágoras se calcula la distancia entre ambos puntos.

$$d(u, v) = \sqrt{\Delta y^2 + \Delta x^2} = R$$

Luego si se sustituye desde (5.6) se tiene:

$$d(a, Cdg) = \sqrt{(y_b - y_a)^2 + (x_b - x_a)^2} = R \tag{5.8}$$

$$= \sqrt{(m_y\lambda)^2 + (m_x\lambda)^2} = R \tag{5.9}$$

$$= \lambda\sqrt{m_y^2 + m_x^2} = R \tag{5.10}$$

De donde se puede despejar λ

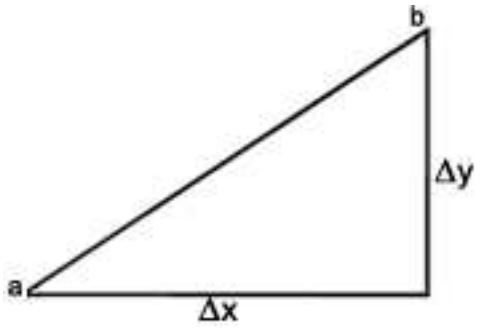


Figura 5.23: Distancia entre dos puntos

$$\begin{aligned}\lambda &= \frac{R}{\sqrt{m_y^2+m_x^2}} \\ &= \frac{R^2}{(m_y\lambda)^2+(m_x\lambda)^2}\end{aligned}$$

La solución para λ es doble, el valor positivo y el negativo de la raíz cuadrada.

Por concordancia entre los signos el valor positivo tendrá sentido para la distancia si el centro de giro se encuentra en el sentido de la rueda derecha, mientras que el negativo tendrá sentido si se encuentra en el sentido de la rueda izquierda.

Con el valor de λ correspondiente, los valores m_x y m_y calculados en (5.7) y la posición del centro entre ruedas se sustituyen en las ecuaciones paramétricas (5.5) y se obtiene x e y del centro de giro.

Si inicialmente el robot tenía una orientación $\theta(t)$ en el instante siguiente el valor de $\theta(t+1)$ será:

$$\theta(t+1) = \theta(t) + \Delta\alpha$$

siendo $\Delta\alpha$ el resultado de (5.3). Adicionalmente se desea que el robot tenga una orientación inicial de $-\frac{\pi}{2}$, por lo que el ángulo sobre el que se calcularán las nuevas posiciones será:

$$\theta(t+1) = \theta(t) + \Delta\alpha + \frac{\pi}{2}$$

Si en el instante t el robot tiene una posición $x_r(t), y_r(t)$ en el instante $(t+1)$ tendrá como posición:

$$\begin{aligned}x_r(t+1) &= R\cos\theta(t+1) + x_{cdg} \\ y_r(t+1) &= R\sin\theta(t+1) + y_{cdg}\end{aligned}$$

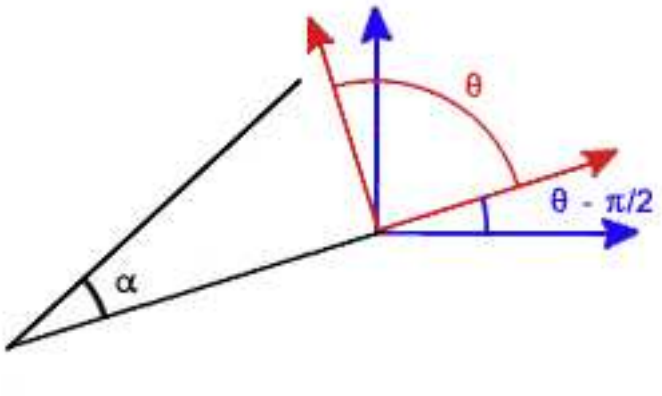


Figura 5.24: Relación entre los ángulos

Todo lo anterior es válido siempre que $\Delta S_d \neq \Delta S_i$. De no ocurrir así, el robot se estará moviendo en línea recta, luego $\Delta\alpha = 0$ (la orientación β no varía respecto del movimiento anterior), y las nuevas posiciones se calcularán por trigonometría simple:

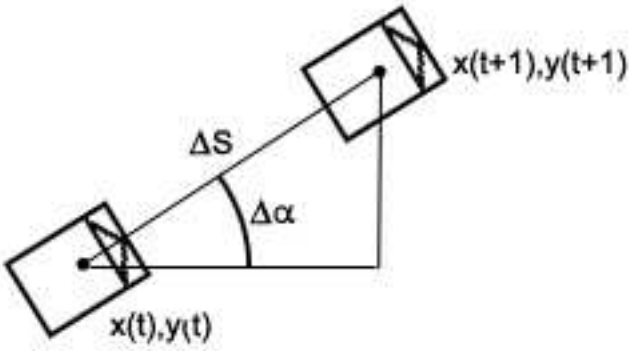


Figura 5.25: Desplazamiento en línea recta.

$$x_r(t+1) = x_r(t) + \Delta S \cos\beta$$
$$y_r(t+1) = y_r(t) + \Delta S \sin\beta$$

Siendo ΔS la traslación del punto medio entre ruedas calculada en (5.4).

5.4 Visualización de imágenes

El sistema teleoperador que se ha creado funciona sobre el sistema gráfico de ventanas X-Window. Sobre este sistema de ventanas es donde se han de visualizar las imágenes

enviadas por el robot. El servidor X-Window puede ser configurado de distintos modos en función de la resolución y el número de colores que se desee utilizar. Los tres modos más utilizados actualmente son: 8 bpp, 16 bpp y 24 bpp. La interfaz gráfica del teleoperador soporta los tres modos. En el modo 8 bpp las imágenes se visualizan en blanco y negro, mientras que en los otros dos modos se visualizan en color.

X-Window es un sistema orientado a displays, que son sistemas de visualización de mapas de bits. Un mapa de bits es una imagen formada por píxels (puntos con un valor numérico asociado que representan un color en el display). Cada píxel tiene una posición y un valor de color asignado. La relación de colores posibles entre los valores de los píxels y los colores reales se conoce como mapa de colores (colormap) y puede ser de dos tipos:

- Tipo paleta, como en el modo 256 colores (8 bpp), donde se dispone de una tabla de tríos RGB (paleta) que especifica a qué color corresponde cada valor numérico del píxel.

- De relación directa, como los modos 16 bpp y 24 bpp, donde cada color se forma a partir de las componentes RGB del color deseado y colocando estas componentes en el lugar correcto dentro de la memoria de video, de forma que representen el color buscado.

Existen diferentes modos de vídeo, es decir, la tarjeta gráfica es capaz de programar el sistema para diferentes resoluciones y número de colores, por ello es necesario detectar el modo de vídeo actual y trabajar con el convenientemente, ya que para trasladar las imágenes de la cámara al *canvas* donde se van a visualizar necesitaremos conocer cómo debemos de representar los píxels de la imagen.

Modo 8 bpp Este modo es el único con paleta. Esta paleta se carga utilizando la función *XallocColor*. A esta función se le pasan unos valores RGB y devuelve la posición en el mapa de colores cuyas componentes de color estén más cercanas a los valores dados. Los valores RGB tendrán que estar dados según la estructura *XColor*, que tiene la definición siguiente:

```
typedef struct{
        unsigned long pixel;
        unsigned short red, green, blue;
        char flags /*do_red, do_green, do_blue*/
```

char pad;

}*XColor;*

Con sucesivas llamadas a *XAllocColor* se rellena la paleta de 256 colores con los valores devueltos por la función. Una vez rellena la paleta de colores se puede calcular la imagen que se desea visualizar desde la imagen original enviada por el robot.

En el modo 8 bpp hay que repartir los 8 bits que ocupan los colores entre las componentes RGB. El patrón de distribución de las componentes se muestra en la figura (5.26).

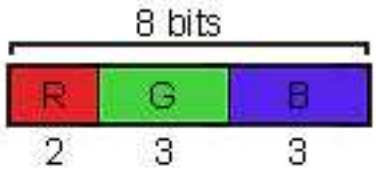


Figura 5.26: Composición del color para el modo 8 bpp

Al componer las imágenes para cada píxel se distribuye un byte siguiendo el patrón anterior, por lo que las operaciones que hay que hacer son:

Obtener los dos primeros bits (los más significativos) de la componente R utilizando una máscara AND con 0xc0. Figuras (5.27) y (5.30).

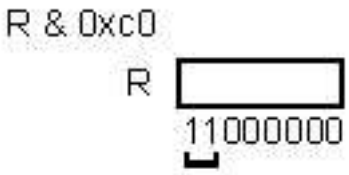


Figura 5.27: Máscara para la componente R en el modo 8 bpp

A la componente G habrá que hacerle una máscara AND con 0xe0 para conseguir sus tres bits más significativos. Figuras (5.28) y (5.30).

Como según el patrón deberá ocupar los tres bits centrales se deberá desplazar este resultado hacia la derecha 2 posiciones (los 2 bits anteriores están ocupados por la componente R).

Para la componente B también hay que obtener sus 3 bits más significativos, por lo que se le hace la misma máscara AND (0xe0), y el resultado se deberá desplazar 5 posiciones a la derecha para que ocupe el lugar adecuado. Figuras (5.29) y (5.30).

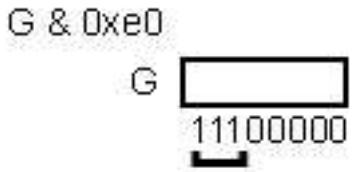


Figura 5.28: Máscara para la componente G en el modo 8 bpp

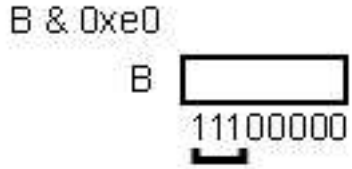


Figura 5.29: Máscara para la componente B en el modo 8 bpp

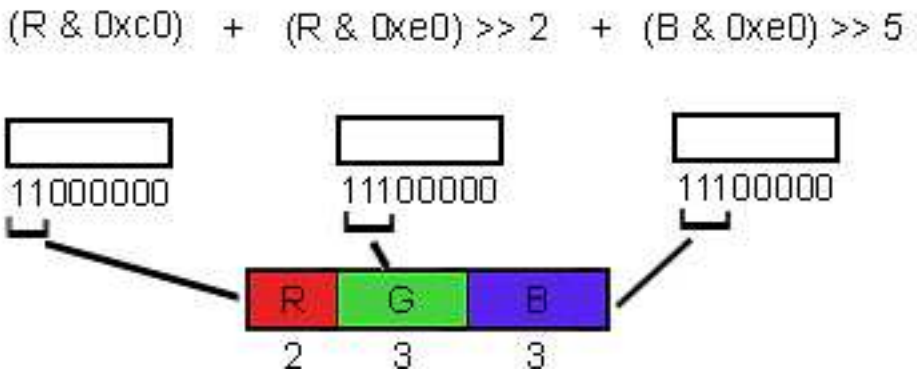


Figura 5.30: Descomposición del color para el modo 8 bits

Los 8 bits conseguidos se corresponden con uno de los 256 valores de la tabla que se rellenó con anterioridad, y que es el color que va a tener el píxel cuando se visualice en pantalla.

Modo 16 bpp En el modo 16 bpp no hay paleta, ya que es un modo de color directo. En este modo sólo hay que calcular la imagen visualizable desde la imagen original. Para calcular la imagen visualizable hay que hacer un reparto de los 16 bits de color para las tres componentes. Para distribuir los colores se utilizará el patrón de color que se muestra en la imagen 5.31.

A la hora de implementar la distribución deberemos dividir los 16 bits en 2 bytes.

Figura 5.31: Composición del color para el modo 16 bpp

El byte 1 se compondrá haciendo la máscara AND 0xf8 a la componente R, para obtener sus 5 bits más significativos. A éstos hay que sumarles el resultado de hacerle a la componente G la máscara AND 0xe0 (así se consiguen sus 3 bits más significativos) y desplazarlos a la derecha 5 posiciones. Figura (5.32).

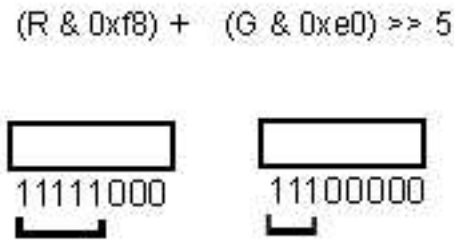


Figura 5.32: Composición para el byte 1 en el modo 16 bpp

El byte 0 se compondrá haciéndole la máscara AND 0xf8 a la componente B (para obtener sus 5 bits más significativos) y desplazando el resultado 3 posiciones hacia la derecha. Al resultado anterior hay que sumarle el obtenido tras hacerle la máscara AND 0x1c a la componente G y desplazarla 3 posiciones hacia la izquierda. Figura (5.33).

Modo 24 bpp El modo 24 bpp también es un modo de color directo. En este modo todo es mucho más sencillo, de manera que no hay que hacer ningún tipo de desplazamiento con las componentes, sino que directamente se asignará un byte para cada una. Figura (5.34).

En los tres modos hay que tener en cuenta que todo lo dicho sirve tanto para las imágenes en escala de grises como en color. Para trabajar en escala de grises lo único que hay que tener en cuenta es que el valor de las componentes RGB de cada píxel será para las tres el mismo, el nivel de gris del píxel.

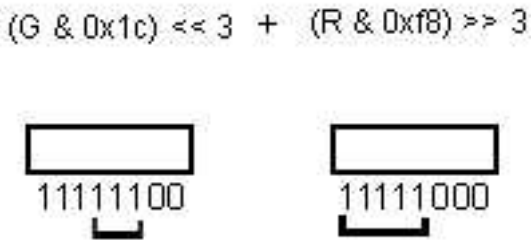


Figura 5.33: Composición para el byte 0 en el modo 16 bpp

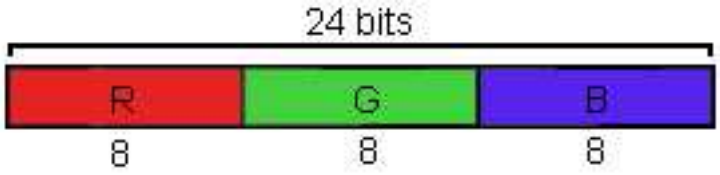


Figura 5.34: Composición del color para el modo 24 bpp

Capítulo 6

Conclusiones y trabajos futuros

Se ha creado un sistema teleoperador para el Eyebot. El sistema permite visualizar en el PC los datos sensoriales del robot (infrarrojos, encoder, cámara) y mover al robot desde el PC utilizando el ratón.

Además este proyecto ha profundizado en el conocimiento del robot Eyebot y ha permitido la familiarización del grupo de robótica con las funciones que ofrece su sistema operativo para acceder a sus sensores y actuadores, utilizar sus sistemas de comunicación y crear programas multitarea. También se han encontrado limitaciones en el robot, que condicionan el rendimiento del teleoperador.

En este capítulo se analiza el teleoperador construido y se proponen líneas de trabajo futuras para superar estas limitaciones y mejorar su funcionamiento.

6.1 Conclusiones

Se ha conseguido implementar un sistema teleoperado con una arquitectura cliente/ servidor, donde el cliente (programa que se ejecuta en el PC) se conecta al servidor (programa que se ejecuta en el Eyebot) a través del puerto serie. El servidor por un lado recoge datos sensoriales del robot y se los envía al cliente y por otro lado recibe órdenes de movimiento que materializa con comandos a los motores del robot. El cliente por su parte recibe los datos que le envía el robot y los visualiza en pantalla para el usuario humano final. Además recoge las órdenes de movimiento generadas por el usuario y las envía al robot.

Se ha desarrollado un *protocolo de comunicaciones* bidireccional entre el PC y el robot que permite tanto el envío de mensajes cortos como de imágenes. Este protocolo permite la suscripción a datos sensoriales (encoders e infrarrojos) que son enviados con mensajes cortos, y el envío bajo demanda de imágenes. El flujo de mensajes de

respuesta a las suscripciones a los infrarrojos y a los encoders no está limitado, y sigue la velocidad de captura de datos sensoriales. El flujo de imágenes está limitado a 0.25 fps.

Tanto el cliente como el servidor han sido diseñados utilizando técnicas de *multiprogramación* que permiten ejecutar distintas hebras en paralelo. En el Eyebot el modo elegido para la multiprogramación entre los que ofrece el sistema operativo es el modo con desalojo, ya que es más robusto frente a bloqueos de las tareas. El sistema operativo del Eyebot ofrece funciones específicas que posibilitan este tipo de multiprogramación. En el PC se ha implementado un programa multihebra utilizando la librería Rai-Scheduler, ya que ésta había sido utilizada dentro del grupo con anterioridad.

A través de una *interfaz gráfica* se visualiza el robot en una pantalla a vista de pájaro, pinchando en unos botones de la interfaz el usuario puede solicitar datos de las lecturas de los sensores del robot. Estos datos quedan representados en la pantalla de una manera gráfica, mediante una representación visual de los mismos. Utilizando 2 botones el usuario puede solicitar imágenes en color o en escala de grises. Estas imágenes se muestran en la interfaz gráfica con la posibilidad de almacenarlas en disco pinchando otro botón. La interfaz gráfica incluye un *joystick* gráfico, a través del cual es posible comandar la traslación y el giro del robot. Adicionalmente unos *diales* permiten controlar la posición de la cámara y del pateador. La velocidad de respuesta de la interfaz ante las modificaciones que en ella se hagan es de 250 milisegundos, suficiente para que el operario tenga la impresión de que hay una respuesta automática del sistema.

En la implementación se han encontrado los siguientes limitaciones:

- Se ha regulado el número de imágenes que se solicitan al robot a 0.25 imágenes por segundo. Las imágenes ocupan muchos bytes en comparación con el resto de mensajes. A pesar de utilizarse la máxima velocidad del puerto serie la transmisión tarda comparativamente mucho tiempo. Si no se controlaran estas peticiones el sistema ocuparía prácticamente todo el tiempo en estas transmisiones y entre una imagen y otra llegarían en bloque los mensajes de respuesta a las suscripciones. Se originaría una discontinuidad en el movimiento del robot, la representación de su estado en el puesto base no se asemejaría a la realidad.

- Hay un flujo máximo de mensajes desde el PC al Eyebot (78 por minuto). Esto

se debe a que por una limitación del sistema operativo del robot se producen errores de lectura al utilizar la cámara a la vez junto con el puerto serie. Para solucionarlo se limita el tiempo de actividad de la cámara. Ésta se inicializa cada vez que se solicita una nueva imagen, y una vez capturada se libera el recurso. Esta limitación obliga a no cerrar el bucle de control de los motores en el cliente. En lugar de eso se tiene una hebra local en el servidor que cierra el bucle de control. Esta hebra admite la modulación velocidad / posición que le envía el cliente cuando hay alguna modificación en el *joystick*.

- La existencia de una unión física (el cable del puerto serie) entre el PC y el robot, limita la libertad de movimiento del Eyebot. No es posible aprovechar al máximo el telemovimiento porque el recorrido del robot está limitado a la longitud del cable.

6.2 Trabajos futuros

1. Para posibilitar una total libertad de movimientos del robot es posible sustituir la comunicación a través del puerto serie por la comunicación a través de un radioenlace. De este modo no existe ninguna conexión física entre el cliente y el servidor, estando entonces la limitación de movimiento condicionada al alcanze del radioenlaze (4m aprox.).

   El inconveniente principal de esta solución es que el envío de imágenes desde el Eyebot al PC sería más crítico aún de lo que es actualmente, ya que la máxima velocidad de transmisión vía radio es de 9600 baudios, doce veces menor que la del puerto serie, que es de 115200 baudios.

2. Una posibilidad abierta es teleoperar al robot en un mundo virtual. La simulación en escenarios virtuales se utiliza para saber cómo se va a comportar el robot durante la ejecución de un determinado programa, utilizando un entorno simulado en lugar del entorno real. El fabricante del Eyebot suministra un simulador llamado EyeSim con una interfaz visual que permite realizar comprobaciones y depurar los distintos programas de una manera fácil.

   Para poder teleoperar un robot en el simulador EyeSim primero hay que cargar un determinado escenario virtual en el simulador. El programa servidor se ejecutaría en el robot simulado, que está dentro de ese escenario, y el programa cliente

debería comunicarse con ese robot simulado. Para ello habría que mejorar el simulador, ya que en estos momentos no da soporte para que los programas que se ejecutan en los robots simulados utilicen el puerto serie.

3. Como ya se comentó en el capítulo 4 la utilización de la librería Rai-Scheduler en lugar de Pthreads se debe principalmente a que el grupo de robótica ya había trabajado anteriormente con ella y conocía su funcionamiento, de modo que su utilización para la implementación del programa cliente resultaba mucho más sencilla que la utilización de Pthreads. El planificador Rai-Scheduler funciona con pseudoparalelismo sin desalojo, donde todas las *tareas RAI* forman parte de un único proceso para el sistema operativo Linux. Este modo de trabajo tiene como principal inconveniente que si se bloquea una tarea las restantes también se detienen. Cuando se trabaja con Pthreads cada tarea del programa cliente se ejecuta en una hebra distinta. El tiempo de CPU es distribuido por el planificador del sistema operativo en el modo con desalojo, de modo que los programas resultantes son mucho más robustos. La librería Pthreads implementa una multitarea real, donde si un proceso se bloquea el resto sigue funcionando. Es por ello que en el futuro es deseable la implementación del programa cliente utilizando Pthreads en lugar de Rai-Scheduler. Además Pthreads viene incluida en las distribuciones recientes de Linux mientras que Rai-Scheduler no.

4. Otra línea de actuación en el futuro debe de encaminarse a la mejora del repertorio de movimientos. Actualmente se tiene un control en posición para el giro y un control en velocidad para la traslación. El objetivo es conseguir un repertorio de movimientos que ofrezca la posibilidad de seleccionar un control en velocidad o en posición, tanto para la traslación como para el giro.

Apéndice A

Anexo-1: Características del sensor GP2D02 de Sharp

GP2D02

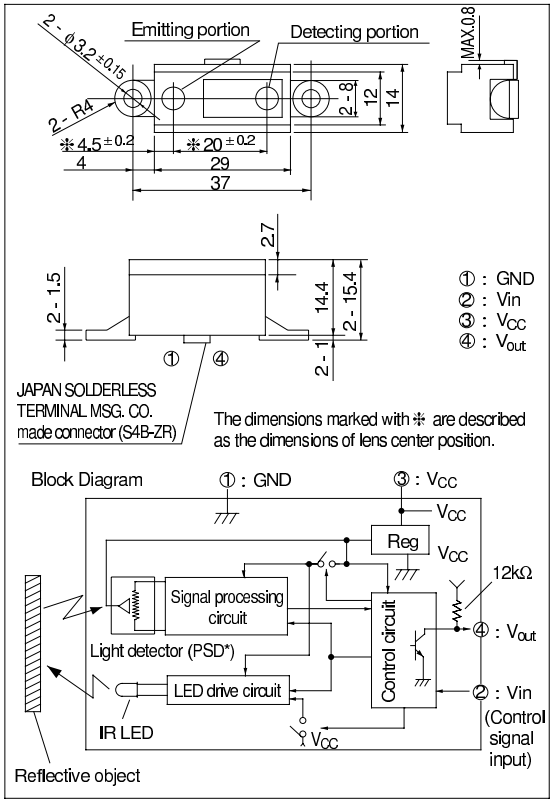
Compact, High Sensitive Distance Measuring Sensor

■ Features

1. Impervious to color and reflectivity of reflective object
2. High precision distance measurement output for direct connection to microcomputer
3. Low dissipation current at OFF-state
   (dissipation current at OFF-state : TYP. 3 μA)
4. Capable of changing of distance measuring range through change the optical portion (lens)

■ Outline Dimensions

(Unit : mm)



■ Applications

1. Sanitary sensors
2. Human body sensors for consumer products such as electric fans and air conditioners
3. Garage sensors

* PSD : Position Sensitive Detector

■ Absolute Maximum Ratings

(Ta=25°C, V_{CC}=5V)

Parameter	Symbol	Rating	Unit
Supply voltage	V_{CC}	- 0.3 to + 10	V
*1Input terminal voltage	V_{in}	- 0.3 to + 3	V
Output terminal voltage	BV_O	- 0.3 to + 10	V
Operating temperature	T_{opr}	- 10 to + 60	°C
Storage temperature	T_{stg}	- 40 to + 70	°C

*1 Open drain operation input

■ Operating Supply Voltage

Symbol	Rating	Unit
V_{CC}	4.4 to 7	V

■ Electro-optical Characteristics

(Ta=25°C,Vcc=5V)

Parameter		Symbol	Conditions		MIN.	TYP.	MAX.	Unit
Distance measuring range		Δ L	*1		10	-	80	cm
Output terminal voltage		V_{OH}	Output voltage at High	L= 20cm	V_{CC} - 0.3	-	-	V
		V_{OL}	Output voltage at Low	*1	-	-	0.3	V
Distance characteristics of output		D	L= 80cm, *1		-	75	-	DEC
		Δ D	Output change at L=80 cm to 20 cm, *1		48	58	68	DEC
Dissipation current	at operating	I_{CC}	L= 20cm, *1, *2		-	22	35	mA
	at OFF-state	I_{off}	L= 20cm, *1		-	3	8	μA
Vin terminal current		I_{vin}	Vin= 0V		-	- 170	- 280	μA

Note) L : Distance to reflective object

DEC : Decimalized value of sensor output (8-bit serial)

*1 Reflective object : White paper (reflectivity : 90%)

*2 Average dissipation current value during distance measuring operation when detecting of input signal, Vin as shown in the timing chart

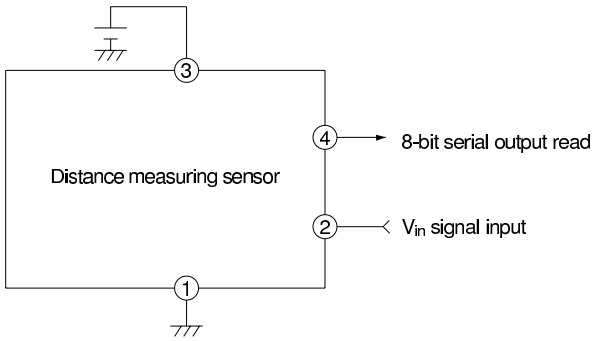
*3 Vin terminal : Open drain drive input.

Conditions : Vin terminal current at Vin OFF-state : -1 μA

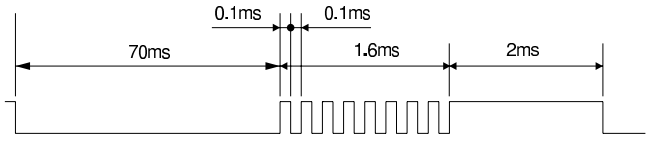
Vin terminal current at Vin ON-state : 0.3V

■ Test Circuit

1. Test circuit



2. Vin input signal for measurement



■ Timing Chart

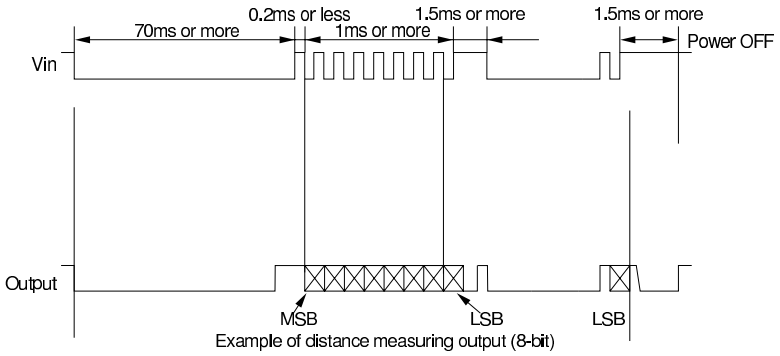


Fig. 1 Distance Measuring Output vs. Distance to Reflective Object

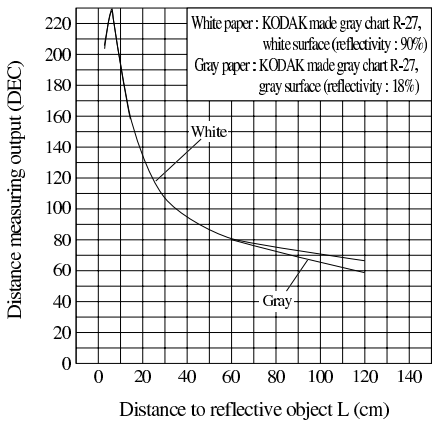
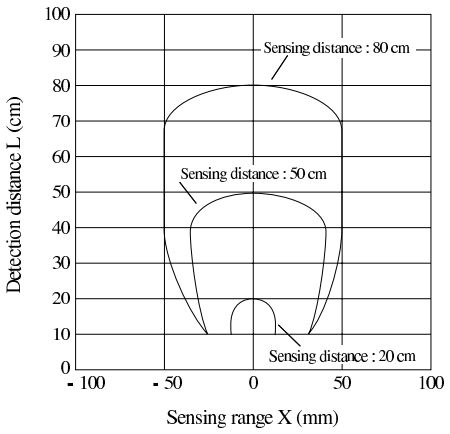


Fig. 2 Detection Distance vs. Sensing Range



Test Method for Sensing Range Characteristics

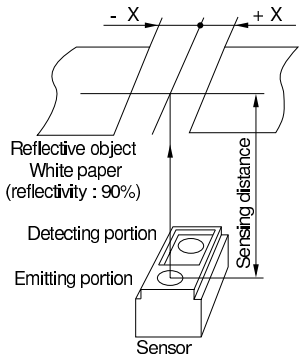
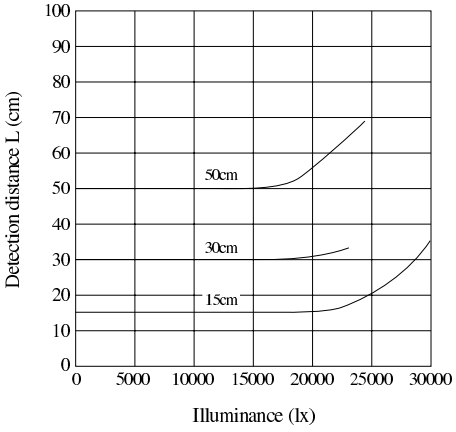
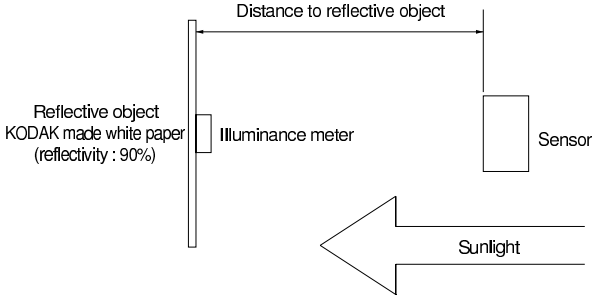


Fig. 3 Detection Distance vs. Illuminance



Test Method for Anti External Disturbing Light Characteristics



Apéndice A

Anexo-2: Código fuente de los programas servidor y cliente

El código fuente de los programas servidor y cliente se encuentra disponible en la dirección de Internet:

http://gsyc.escet.urjc.es/~jmplaza/grupo.robotica/eyebot.html

Bibliografía

[1] Antonio Barrientos, Carlos Balaguer, Luis Felipe Peñín, Rafael Aracil. *Fundamentos de robótica.* McGrawHill, 1997.

[2] Brian W. Kernighan, Dennis M. Ritchie. *El lenguaje de programación C.* Prentice Hall, 1991.

[3] C. Vázquez Regueiro, J.M. Cañas, M.C.García-Alegre. *Real time visualization of robot-environment states in local piloting.* Informe Técnico, Instituto de Automática Industrial (CSIC), Marzo 1997.

[4] Jose María Cañas. *Interfaz gráfica para el controlador de un robot móvil.* Informe Técnico, Instituto de Automática Industrial (CSIC), Julio 2001.

[5] Esther García, Jose María Cañas, Vicente Matellán. *Manual del robot Eyebot.* Informe Técnico, Univ. Rey Juan Carlos, Enero 2002.

[6] Leandro Fernández García. *Desarrollo de una interfaz gráfica para el control teleoperado del robot escalador ROMA.* Proyecto fin de carrera, Universidad Carlos III de Madrid, 2000.

[7] Iberdrola. *PROYECTO SRT - Robot Móvil Guiado por Carril.* http://www.iberdrola.es/conozca/generaenergia/destecnologico/proyterminado.htm.

[8] J. O'Sullivan, G.D. Armstrong, Karen Zita Haigh. *Xavier- The manual v0.4.* Robot Learning Laboratory, Computer Science Department, Carnegie Mellon University, Abril 1997.

[9] Phillip John McKerrow. *Introduction to robotics.* Addison-Wesley, 1991.

[10] José María Cañas Plaza. *Manual del robot móvil Hermes.* Informe Técnico, Instituto de Automática Industrial (CSIC), Julio 2001.

[11] Real Worl Interface (RWI). *Inc. System software and RAI-1.2.2 documentation.* http://www.rwii.com, 1996.

[12] T.C. Zhao, Mark Overmars. *Forms Library: a graphical user interface toolkit for X.* http://world.std.com/ xforms/.

[13] Thomas Braunl, The Univ. of Western Australia. *Eyebot Documentation.* http://www.ee.uwa.edu.au/~braunl/eyebot.

[14] X.Org. *X-Window System.* http://www.x.org.