UNIVERSIDAD REY JUAN CARLOS

Ingeniería Técnica en Informática de Sistemas

Escuela Superior de Ciencias Experimentales y Tecnología

Curso académico 2001-2002

Proyecto Fin de Carrera

Comportamiento sigue pelota en un robot con visión local

Tutor: José María Cañas Plaza

Autor: Félix San Martín de la Fuente

Septiembre 2.002

A mi familia y amigos

Agradecimientos

Quiero dar las gracias de manera especial a Jose Mª Cañas y a Vicente Matellán por haberme facilitado sus conocimientos sobre redes y robótica, y a los demás miembros del grupo de robótica de la universidad.

Índice general

Índice de cuadros

Índice de figuras

Resumen

EyeBot es un robot móvil sobre el que estamos trabajando en el grupo de robótica de la Universidad Rey Juan Carlos. Dispone de microprocesador Motorola 68332 y una serie de sensores y actuadores que le permiten desenvolverse por el entorno, de entre ellos una cámara, dos motores y dos servos. El sistema operativo del robot incorpora una consola a través de la cual se puede realizar chequeos de los distintos componentes hardware y se pueden descargar programas al robot. Además ofrece una interfaz de programación mediante la cual los programas de usuario pueden acceder a los distintos dispositivos.

Este proyecto desarrolla en el EyeBot el comportamiento sigue-pelota que busca mantenerse centrado respecto a la pelota, a una cierta distancia. Si la pelota avanza el robot debe perseguirla para que no se aleje demasiado, si está demasiado cerca debe retroceder. Cuando la pelota se mueve a la derecha del robot éste debe girar hacia ese lado y de modo contrario si la pelota se mueve hacia la izquierda, para mantenerse centrado. Para conseguir este comportamiento, el robot captura imágenes con su cámara, y en ellas reconoce la pelota y su posición relativa. El robot se mueve dependiendo de dónde esté situada la pelota en la imagen. Por ejemplo si la pelota está escorada hacia la derecha el robot gira en ese mismo sentido para centrarla en su imagen, si está a la izquierda girará hacia la izquierda. Si la pelota aparece en la parte superior de la imagen el robot avanza y si aparece por debajo retrocede. Disponemos de dos controles simultáneos, un control en X, derecha e izquierda, y un control en Y, adelante y atrás.

El proyecto ha sido desarrollado sobre una plataforma Linux, utilizando las herramientas que se proporcionan con este sistema operativo, desde el editor para escribir el programa, hasta el procesador de documentos para hacer la memoria pasando por el compilador cruzado necesario para compilar los programas que se ejecutan en el robot.

Capítulo 1

Introducción

Los robots aparecieron a mitad del siglo XX. Hace 30 años lo que hoy conocemos por robot era algo propio de la ciencia-ficción. Los primeros avances en este campo fueron dificultosos y tímidos. Tuvo su gran áuge en la década de los 80 donde ya empezaban a ser utilizados en las fábricas para realizar las tareas repetitivas, llegando a la madurez, según algunos autores, en la década de los 90, dónde ya es normal verlos en cualquier tipo de industria.

Con el objetivo de diseñar una máquina flexible, adaptable al entorno y de fácil manejo, George Devol, pionero de la Robótica Industrial, patentó en 1948, un manipulador programable que fue el germen del robot industrial. Los primeros robots eran brazos mecánicos teleoperados, manejados por un operador careciendo de autonomía, con pocos grados de movimiento para hacer más fácil su manejo. Fueron estos telemanipuladores los que dieron paso al robot, cambiaron al operador por un ordenador que controlaba los movimientos del brazo. Estos estaban principalmente pensados para realizar tareas repetitivas, por ejemplo, los robots soldadores encargados de realizar siempre las mismas soldaduras en la industria del automóvil, y hostiles, son usados por la industria nuclear para proteger al operador de la radiactividad.[Aracil97]

El primer campo donde se introduccieron los robots fue en la industria. La definición de robot industrial más comúnmente aceptada posiblemente sea la de la Asociación de Industrias Robóticas (RIA), según la cual un robot industrial es un manipulador multifuncional reprogramable, capaz de mover materias, piezas, herramientas, o dispositivos especiales, según trayectorias variables, programadas para realizar tareas diversas. En la década de los 70 la industria espacial se sumó a este interés por los robots.

El estudio de la robótica en la universidad ha sido pionera en el gran avance de los robots en la industria. Las investigaciones llevadas a cabo en la universidad han sido

llevadas a la práctica en las industrias.

En nuestros días se ha producido un gran avance en el desarrollo de los robots móviles, que son aquellos que disponen de algún mecanismo, patas, ruedas, que les permiten moverse por el entorno. Estos robots disponen de una mayor libertad de movimiento pudiendo realizar tareas de limpieza, de guías en museos, viajar por el espacio, etc.

La robótica que está entrando en nuestras vidas es la robótica de compañía[Matellan02] emulando a mascotas, como el AIBO de sony[1]. Quién iba a pensar hace unos años que su perro sería un robot que respondería a la voz, a las caricias. Esto era totalmente impensable por nuestros abuelos, pero hoy, gracias a los avances tecnológicos, además podemos montar en casa nuestro propio robot con, por ejemplo, los kit robóticos de LEGO, o que nuestro guia en un museo sea un robot.

En la universidad Carnegie Mellon se ha desarrollado el robot FLO, con cara de aspecto humano, al cual le puedes hacer preguntas. Por ejemplo, qué ponen en la tele o qué tiempo va a hacer.

La Robótica es una tecnología multidisciplinaria, ya que ésta hace uso de los recursos que le proporcionan otras ciencias afines, solamente hay que pensar que en el proceso de diseño y construcción de un robot intervienen muchos campos pertenecientes a otras ramas de la ciencia, como pueden ser la mecánica, la electrónica, la informática, la teoría de autómatas y de control.

La combinación de todas las disciplinas y otras muchas, más el conocimiento de la aplicación a la que se enfoca, forman la robótica, por lo que su estudio se hace especialmente indicado en las carreras de Ingeniería Superior y Técnica y en los centros de Formación Profesional. La Robótica brinda a investigadores y aficionados un vasto y variado campo de trabajo, lleno de objetivos y en estado inicial de desarrollo.

1.1. Robocup

La RoboCup[2] es una iniciativa de educación e investigación internacionales, un intento de promover la investigación en la Inteligencia Artificial(IA). Está encaminada al campo de los robots inteligentes, tienen que resolver un problema, donde diversas tecnologías pueden ser integradas y examinadas.

Para este propósito, Robocup eligió el fútbol como problema a resolver, y empezó a

[1] http://www.aibo.com
[2] http://www.robocup.org

organizar campeonatos de fútbol entre robots, alrededor de estos campeonatos también se dan conferencias, exposiciones. La primera celebración oficial de la Robocup fue en el año 1997.

Para crear un equipo de robot que pueda jugar al fútbol se necesita la aplicación de diversas tecnologías como agentes autónomos, multi-colaboración de agentes, adquisición de estrategias, principios de razonamiento en tiempo real, combinación de sensores entre otras. Robocup es una tarea para un equipo de múltiples robots bajo un entorno dinámico.

Actualmente, Robocup tiene:

- Liga de Robot: Cada equipo consta de 5 robots pequeños. En esta competición se dispone de una cámara cenital que proporciona información a ambos equipos. El campo de juego es una mesa de ping pong de color verde, la pelota es de golf y de color naranja, y cada robot cuenta con etiquetas identificativas de colores. Además estos tienen un tamaño limitado por la organización, el robot tiene que entrar en un círculo de 180mm de diámetro. El procesado de las imágenes y las tácticas de juego se realizan en un ordenador separado debido a la poca capacidad de cómputo que tienen estos robots, luego esta información hay que enviarla a los robots. El juego de estos robots es totalmente autónomo. Un ejemplo de esta competición lo podemos ver en la figura 1.1. Hay otra variante con 11 robots por equipo (TBA)

Figura 1.1: Competición liga pequeña

- Liga de Simulación Pequeña (f-180): Liga para equipos de 11 jugadores que funcionan en un simulador.

- Liga de Robot de Tamaño Medio (f-2000): Consta de 5 robots por equipo pero de un tamaño más grande. Cada robot dispone de una cámara local y todo el procesamiento de su comportamiento se realiza localmente en el ordenador de abordo.

- Liga de Robot con Patas (Patrocinado por Sony): Los equipos están formados por aibos[3], robots con aspecto de perrito, de sony.

- Liga de humanoides (Desde 2002): Nuevo en la última edición. Consiste en el lanzamiento de penaltis entre robots con aspecto de personas.

Además también de disputa una liga de robots de salvamento y una liga de simulación de salvamento

1.2. Grupo robótica URJC

El grupo de robótica[4] de la URJC surgió en el año 2000. En un primer momento era un grupo reducido de alumnos y profesores de la universidad contando con algún miembro ajeno a esta. El grupo tiene varias lineas de investigación, los robots de Lego, la Robocup, robots de interiores,etc.

El grupo de robótica de esta universidad se planteó la creación de un equipo de fútbol para la participación en una competición internacional entre robots, la Robocup. Primero había que decidir que tipo de robots se iba a utilizar, ya que para participar en esta competición deben de tener unas dimensiones reducidas. Después de hacer varias comparativas entre robots muy parecidos se inclinaron a favor del EyeBot, principalmente porque dispone de cámara local y su precio no es excesivamente elevado. Los robots del equipo se pueden ver en la figura 1.2.

Además este robot esta pensado para jugar al fútbol, ya que cuenta con un pateador para dar a la pelota. Incorpora tres sensores infrarrojos para detectar obstáculos cercanos, dos encoders para medir el desplazamiento realizado y una cámara.

Dentro de la Robocup también queremos participar en la liga de simulación con un equipo cuyo control está basado en lógica borrosa. Este equipo es el resultado del proyecto fin de carrera de un componente del grupo[Alvarez01].

[3]http://www.aibo.com
[4]http://gsyc.escet.urjc.es/robotica

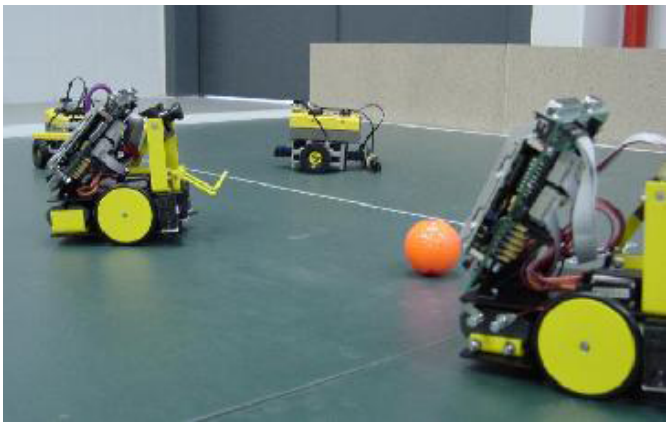

Figura 1.2: Robots de la universidad

Para la construcción del equipo de robot debemos indagar en todos los aspectos que conlleva la competición, es decir, para hacer que los robots jueguen al fútbol debemos saber como comunicar unos robots con otros, comunicarnos con el PC, manejar apropiadamente los motores y la cámara.

Por ello nosotros estamos probando las posibilidades que tiene estos robots, las cosas que podemos hacer con ellos y cuales resultan imposibles.

Para conocer mejor las posibilidades del robot que elegimos, el EyeBot, se realizó un proyecto que consistía en la creación de un teleoperador[García01] para este robot. El cual permitía mover el robot, capturar imágenes, localizar obstáculos. Nos permitió saber más acerca del funcionamiento de los distintos elementos de los que dispone el EyeBot.

Una vez que hemos aprendido a manejar el robot y ver que posibilidades nos ofrece, ya sea haciendo aplicaciones sencillas que usen los distintos componentes del robot o mediante un teleoperador, vamos a hacer que sea capaz de reconocer objetos, seguir paredes[Victor02].

Otra de las líneas del grupo es la generación de comportamientos autónomos en robots de interiores, como pueden ser guias de un museo. Ver figura 1.3. También se investigan algoritmos de percepción complejos así como diferentes estructuras de control para producir el comportamiento deseado.

En el grupo se intenta trabajar dentro de la arquitectura JDE (Jerarquía Dinámica de Esquemas)[Cañas02]. La idea básica de JDE es la descomposición del comportamiento en esquemas perceptivos y esquemas motores. Los perceptivos se encargan de proporcionar información que puede ser leída por otros esquemas, y los esquemas mo-

Figura 1.3: Robot Pioneer para interiores

tores usan esta información para realizar las acciones necesarias. Los esquemas son activables a voluntad y se organizan en jerarquías que se crean dinámicamente.

En esta arquitectura estos esquemas están organizados en una jerarquía dinámica, esto es, si un esquema motor necesita unos datos activa a los esquemas perceptivos correspondientes. Estos a su vez pueden activar otros esquemas, ya sean perceptivos o motores. Así se va creando una jerarquía de esquemas.

1.3. Comportamiento sigue pelota

El presente proyecto trata de conseguir que el robot sea capaz de seguir una pelota. El comportamiento sigue-pelota busca mantener centrado el robot respecto a la pelota, a una cierta distancia. Si la pelota avanza el robot debe avanzar hacia ella, si está demasiado cerca debe retroceder. Cuando la pelota se mueve a la derecha del robot éste debe girar hacia ese lado y de modo contrario si la pelota se mueve hacia la izquierda, para mantenerse centrado. Si en algún momento el robot no ve la pelota no la buscará, se quedará parado. Además la posición de la cámara irá fijada manualmente, en el comportamiento no se mueve la cámara.

Siguiendo las líneas generales del grupo, el proyecto se sitúa entre dos líneas de investigación. Una línea es el profundizar en el conocimiento del EyeBot y en sus posibilidades para comportamientos autónomos aprovechando visión local. la otra linea consiste en desarrollar un comportamiento autónomo sencillo, y a la vez útil, dentro del entorno de la Robocup. Además el proyecto está enmarcado dentro de la arquitectura JDE propuesta por el grupo.

Esta memoria se articula en 5 capítulos. En el segundo capítulo planteamos los objetivos que el presente proyecto debe cubrir, es decir, el comportamiento autónomo sigue pelota. En el tercero se describe el robot concreto para el que se ha diseñado el PFC, tanto sus dispositivos hardware como la interfaz de programación para manejarlos desde programa. También se hace una breve descripción de lo que necesitas para compilar los programas y como descargarlos al robot. En el capítulo 4 describimos con detalle la arquitectura software del programa desarrollado y los diferentes algoritmos empleados para identificar y caracterizar la pelota, comandar movimientos al robot,etc.

Finalizamos la memoria con el capítulo 5 donde presentamos las conclusiones del trabajo y las líneas futuras en las que se puede mejorar, así como los objetivos cubiertos.

Capítulo 2

Objetivos

A lo largo de este capítulo vamos a hacer una descripción del problema que tratamos de resolver, los objetivos concretos que tiene este proyecto y los requisitos que condicionan el mismo. Así como de los distintos campos de conocimiento en los que hemos investigado para resolverlo.

2.1. Descripción del problema

El principal objetivo de este proyecto es conseguir que nuestro robot, el EyeBot, sea capaz de realizar el comportamiento "sigue-pelota". Además queremos profundizar en el conocimiento del EyeBot y en su uso, con vistas a incorporarlo como jugador de la liga pequeña de la Robocup.

El comportamiento "sigue-pelota" consiste en reconocer y seguir una pelota. El objetivo es situarse enfrente de la pelota, si la pelota está a la derecha girará hacia la derecha, si está a la izquierda girará a la izquierda. Si la pelota está lejos del robot, éste avanzará y si está demasiado cerca retrocederá. Para conseguir este comportamiento, el robot captura imágenes con la cámara que lleva incorporada, analizando esta imagen reconoce la pelota, para su posterior seguimiento. El robot se mueve dependiendo de dónde esté situada la pelota en la imagen. Por ejemplo si la pelota está escorada hacia la derecha de la imagen el robot gira en ese mismo sentido para centrarla en su imagen, si está a la izquierda girará hacia la izquierda. Si la pelota aparece en la parte superior de la imagen el robot avanza y si aparece por debajo retrocede. Disponemos de dos controles simultáneos, un control en giro, derecha e izquierda, y un control en tracción, adelante y atrás. La pelota elegida es la que utiliza la Robocup en sus competiciones, una pelota de golf de color naranja.

La velocidad a la que seguirá la pelota no será muy alta ya que estamos limitados

por la capacidad de procesado del robot y por la velocidad de captura de imágenes.

Uno de los objetivos concreto de este proyecto es el diseño de la arquitectura de control siguiendo las lineas de la arquitectura JDE con la que trabaja el grupo de robótica de la URJC.

Se propone la separación de la percepción y del control en esquemas específicos: esquema perceptivo y esquema motriz. En el esquema perceptivo captura las imágenes y las analiza para obtener los datos necesarios para mover el robot. En la parte motora, a partir de unos datos proporcionados por la parte anterior, debemos mover el robot para seguir la pelota y conseguir situarnos enfrente de ella. Este diseño esta orientado a generar el comportamiento deseado en el robot.

Otro objetivo concreto el la verificación de este diseño desarrollando una implementación en el robot real.

Para la implementación de este diseño vamos a explorar y desarrollar unas técnicas de filtrado y segmentación (tratamiento de imágenes) que permitan identificar y caracterizar la pelota. El objetivo de estas técnicas de percepción es reconocer todo aquello que es de color naranja en la imagen, mediante el filtrado, y a partir de esto obtener el estímulo pelota, mediante el segmentado de la imagen para distinguir entre sí las distintas zonas naranjas.

Necesitamos el desarrollo de unas técnicas de control que nos permitan mover el robot adecuadamente para que persiga la pelota. Vamos a desarrollar y evaluar la viabilidad de dos modos de control, control en posición y control en velocidad, para ver cual se ajusta más al objetivo principal de este proyecto.

En el control en posición el robot se mueve cierta distancia tanto en giro como en tracción hasta que consigue centrar la pelota. En el control en velocidad el robot se mueve a una cierta velocidad, dependiendo de donde esté situada la pelota, de tal forma que irá rápido cuando esté lejos hasta pararse cuando la pelota esté enfrente del robot a la distancia deseada.

En este proyecto si la pelota se sale de la imagen el robot no la busca debe pararse, solo consiste en seguir la pelota en la medida de lo posible. Si la pelota va demasiado rápido se pierde. Tampoco mira la presencia de otros obstáculos ni utiliza los infrarrojos.

Tanto el servo de la cámara como del pateador del EyeBot no se utilizan por el programa. La cámara ha de estar centrada y su ángulo de inclinación se fijará manual-

mente.

2.2. Requisitos

Principalmente necesitamos cumplir tres requisitos, que se ejecute en el robot EyeBot, esté encuadrado dentro de la arquitectura JDE y se realice en tiempo real.

Nuestro robot, EyeBot, tiene dos motores, uno a cada lado del robot que le permiten desplazarse por el entorno y girar sobre si mismo. Esto nos resulta muy útil para realizar los giros. La cámara debe poder capturar imágenes en color ya que sino no podríamos hacer un segmentado de la imagen para reconocer la pelota naranja.

Este robot nos ofrece una funcionalidad pero también impone unas restricciones ya que dispone de poca memoria, obligándonos a hacer programas pequeños, y un procesador "lento" (comparado con un PC), teniendo que hacer algoritmos rápidos y eficientes.

Otro requisito es que el diseño del programa debe seguir la filosofía de la arquitectura JDE. Las dos partes mencionadas anteriormente se pueden paralelizar siendo cada una un esquema diferente que comparten algunos valores. Esto nos lleva a hacer el programa multihebra introduciendo conceptos de multiprogramación.

Se puede materializar las ideas de JDE en la plataforma EyeBot ya que dispone de soporte para hebras necesarias en la paralelización.

Otro requisito importante es que el análisis de la imagen y el consiguiente movimiento del robot han de ser en tiempo real. Esta factor de vivacidad nos limita mucho el tiempo al analizar la imagen, sobre todo al filtrado del color naranja, debido a la poca capacidad de cómputo del robot. Este, al no disponer de coprocesador matemático, las operaciones con variables reales(punto flotante) las tiene que realizar el mismo procesador, mediante una emulación, produciendo un retardo considerable en la fase de análisis. Esto nos obliga a utilizar algoritmos eficientes, tanto en consumo de memoria como en uso del procesador.

Para que el robot sea capaz de seguir la pelota en tiempo real es necesario que el análisis de las imágenes sea vivaz, esto es que proporcione las suficientes imágenes por segundo para ser capaces de seguir la pelota en tiempo real. Se estima que ritmos menores a 2 fps (frames per second, en español, imágenes por segundo) serán insuficientes.

El proyecto está realizado sobre una plataforma Linux y desarrollada en ANSI-C, ya que las máquinas del grupo funcionan con este sistema operativo y disponemos de un compilador cruzado de C para el EyeBot.

Capítulo 3

EyeBot

En este capítulo se hace una descripción del robot que usamos para nuestras investigaciones entorno a la Robocup, tanto en sus elementos físicos, sensores y actuadores, como en el software que permite acceder a ellos y manipularlos desde un programa. La descripción es general pero hacemos hincapié en los elementos que usamos para este proyecto, como son la cámara y los motores principalmente.

El EyeBot es un robot comercial. En la web del fabricante, el Profesor Braunl de la Universidad de Australia[1], se puede obtener información acerca de este robot. O puedes adquirir uno en la web del distribuidor europeo[2].

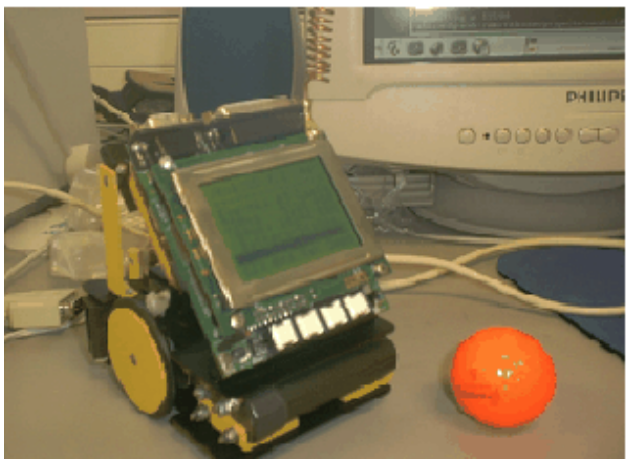

Figura 3.1: EyeBot

Este robot nació en el entorno universitario y ha sido utilizado en muchos proyectos, por ejemplo ha sido utilizado con éxito en la competición de fútbol robótico como muestra la figura 3.2.

[1]http://www.ee.uwa.edu.au/ braunl/eyebot
[2]http://www.joker-robotics.com/

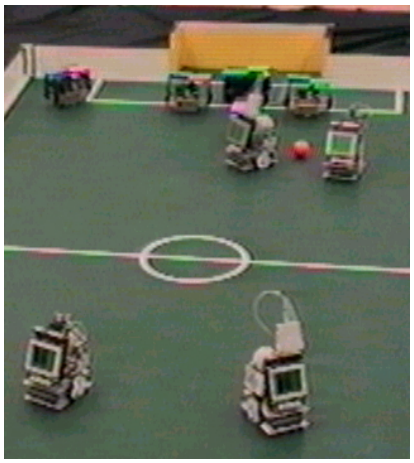

Figura 3.2: EyeBots jugando al fútbol

Tiene cierta ventaja competitiva frente a otros robots similares, como Kephera, Amigo, Lego, debido principalmente a la cámara que lleva incorporada y que los otros no tienen. Además su precio no es muy elevado, está sobre los 1660 Euros.

El robot está constituido por un microprocesador, varios sensores, varios actuadores, elementos de interacción y de comunicación. El microprocesador es el encargado de ejecutar los distintos programas para el robot. La 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 el código de usuario, tiene una memoria RAM de 1 MB que permite ejecutar los programas almacenados en la ROM.

Los sensores recogen información del 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. La cámara del EyeBot permite capturar imágenes en color y en blanco y negro.

Los actuadores permiten al robot desenvolverse en dicho entorno. Dispone de distintos tipos de actuadores, los motores, que permiten el desplazamiento del mismo por el entorno, y los servos, que permiten mover la cámara y el pateador.

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.

El robot dispone de tres posibilidades para establecer una comunicación con el exterior, ya sea con otro robot, o con un PC. Se trata de la comunicación vía puerto

serie, y a través del puerto paralelo. En ambas se han de establecer a través de un cable que una ambas partes. La tercera opción es una comunicación sin cables, utilizando el módulo de radio. A través de un conector RS-232 situado en la parte frontal del robot es posible conectar el EyeBot a un PC. De este modo se posibilita la descarga de programas o el envío de comandos a través del puerto serie. El puerto paralelo se utiliza para conectar el EyeBot con el ordenador y poder realizar una depuración del programa.

En el resto del capítulo ampliaremos la descripción de todos los recursos hardware del robot, que se pueden agrupar en varias categorías. Además veremos como se accede a ellos a través del interfaz de programación proporcionado por el sistema operativo, haciendo hincapié en el manejo de los motores y la utilización de la cámara. Para una descripción más detallada del robot y su programación ver el manual[Cañas01].

3.1. 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.

En esta sección describiremos los dos primeros, mientras que el resto del capítulo explicará con detalle las funciones del API. Para facilitar el acceso a los distintos recursos desde los programas del usuario RoBIOS ofrece una interfaz de programación.

3.1.1. 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 de ejecutarlos.

Tests internos Se pueden realizar chequeos de los distintos elementos del robot mediante esta consola. Se pueden realizar pruebas de funcionalidad a los motores, a los sensores, a los servos, etc.

Carga y ejecución de programas. También es posible configurar los distintos puertos del sistema en cuanto a la velocidad y el modo de transmisión.

Figura 3.3: Consola del EyeBot

Por ejemplo para cambiar la velocidad de transmisión por el puerto serie en el robot hay que 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.

Ejecución de demos. El fabricante proporciona tanto el ejecutable como el código fuente de este tipo de programas.

Ejecución de programas Para la ejecución de un programa este ha debido ser descargado previamente y guardado en la memoria RAM del EyeBot.

Además se pueden almacenar hasta tres programas en la memoria ROM de que dispone el robot y de aquí podemos copiar el que queremos ejecutar en la RAM, como muestra la figura 3.4.

3.1.2. HDT

A un robot se le pueden conectar distintos dispositivos, por ejemplo un servo nuevo para mover otro elemento del robot, otro motor para ir mas rápido. Cuando se quiera añadir un nuevo hardware se ha de modificar la HDT para indicar al sistema operativo este cambio.

El HDT (Hardware Description Table) es una tabla que permite a los controladores hardware detectar y usar de una manera relativamente sencilla el hardware conectado al EyeBot. Consta de dos partes: procedimientos de acceso y estructuras de datos.

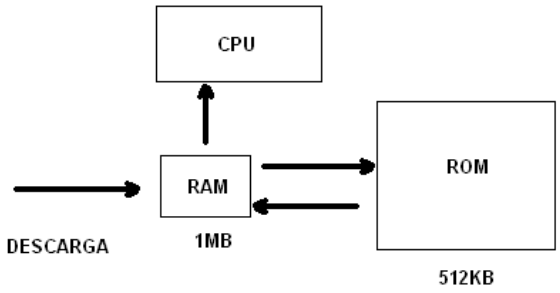

Figura 3.4: Interacción memorias RAM-ROM-CPU

Esta tabla se encuentra dentro del sistema operativo y en ella se definen los diferentes dispositivos hardware que se pueden conectar al EyeBot. Proporciona el soporte software para el distinto hardware conectado y permite reconfigurar el sistema para poder conectar distintos actuadores y sensores a la placa base.

Figura 3.5: Tipos de EyeBot

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.

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.

- Procedimientos de acceso.
  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.

- Estructuras de datos.
  Contiene toda la información sobre el hardware del que dispone el EyeBot y de cómo acceder al mismo.

3.2. Sensores

El robot dispone de tres clases de sensores: los encoders, los infrarrojos y la cámara. Los dos últimos recogen información del entorno para su posterior tratamiento. Los encoders informan sobre la posición y orientación del robot en dicho entorno, cuantificando el desplazamiento que ha realizado cada rueda.

- Infrarrojos
  Los sensores de infrarrojos (o PSD, Position Sensitive Detector) 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. Según sus hojas de características su rango va de los 10 a los 80 cm. Nuestro robot viene equipado con tres infrarrojos, situados en su frente, costado derecho y costado izquierdo.

  Para acceder por programa a los sensores de infrarrojos únicamente requiere la inicialización previa de los mismos con la función `PSDInit(nombre_psd_en_HDT)` que devuelve un manejador. Una vez hecho esto es posible obtener los valores medidos por cada uno de ellos a través de la función `PSDGet(manejador)`.

- Encoders o cuenta vueltas
  El robot tiene dos encoders, situado uno en cada motor. Cada uno devuelve un número de pulsos que refleja el ángulo girado por cada rueda, y con ello, este valor indica el desplazamiento que cada rueda ha realizado.

  Para acceder a los encoders es necesaria su inicialización previa con la función `QuadInit(nombre_encoder_en_HDT)` que entrega un manejador, tras lo cual será posible inicializarlos (`QuadReset(manejador)`) y realizar lecturas de los mismos con la función `QuadRead(manejador)`.

  Es conveniente liberar el manejador en la finalización del programa a través de la función `QuadRelease(manejador)`.

- 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. Las funciones de acceso básico a la cámara se resumen en la tabla 3.1

  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), donde WIDE es disminución del zoom, NORMAL es zoom normal, como lo ve la cámara, y TELE es aumento del zoom.

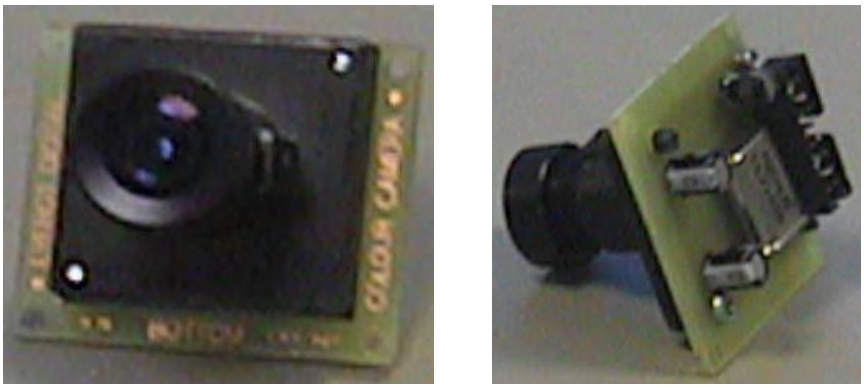

Figura 3.6: Cámara del EyeBot

  Para obtener imágenes en escala de grises se utiliza la función `CAMGetFrame(imagen)` o imágenes en color usando la función `CAMGetColFrame(imagencolor)`.

  La cámara dispone de auto-iris, que funciona por defecto. De hecho 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 hasta que se estabiliza y qué tipo de brillo haya sido seleccionado, ya que dispone de una función que selecciona el tipo de ajuste del brillo, automático(AUTOBRIGHTNESS) o manual(NO AUTOBRIGHTNESS), es la función `CAMMode(modo)`. La máxima velocidad de captura es de 4 imágenes por segundo con imágenes en color.

  La cámara no dispone de un autofocus por lo que tiene que ser ajustada manualmente girando el objetivo.

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

Además de ésta interfaz básica proporcionada por el sistema operativo también el fabricante proporciona otra librería útil para el procesamiento de imágenes. Por ejemplo es posible aplicarles distintos filtros. 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)`).

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.

Cuadro 3.1: Funciones para la cámara

3.3. Actuadores

- Motores de continua

  Dos motores de continua posibilitan el movimiento del robot. Esta movilidad es de tipo tanque, las ruedas directrices no cambian de orientación(como sí lo hacen en los coches). Esto significa que para realizar los cambios de dirección, o giros, es preciso que la rueda correspondiente se desplace a mayor velocidad que la contraria. Por ejemplo, si queremos girar a la derecha la rueda izquierda deberá girar mas deprisa que la rueda derecha.

  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.

Los motores llevan asociados un cuenta vueltas, o encoders, para cada uno, lo que permite poder realizar realimentación. Por ejemplo, se puede establecer un control en velocidad, control en lazo cerrado, que admite como consigna la velocidad objetivo que se va comparando con la velocidad medida a través de los encoders.

Hay dos formas de programar los motores, bien usando la interfaz directa del sistema operativo(lazo abierto) o bien usando la librería VW proporcionada por el fabricante del robot(lazo cerrado).

Para la interfaz básica proporcionada por el sistema operativo, es necesaria una inicialización utilizando la función `MOTORInit(nombre_motor_en_HDT)` que devuelve un manejador. Tras esto se puede indicar al motor deseado que se desplace con la función `MOTORDrive(manejador,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 los movimientos resultantes no serán muy precisos porque no lleva realimentación.

Existe otro modo de mover los motores que utiliza un control en lazo cerrado. Con este tipo de control realimentado se obtienen unos movimientos mucho más precisos. Las funciones de ese control se agrupan en la librería VW.

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 señal que suministra a los motores. De este modo el control del movimiento se realiza con una precisión muy superior a la conseguida a través del movimiento de los motores de manera directa.

El uso de esta librería requiere de una inicialización a través de la función `VWInit(nombre_en_HDT,escala_actualización)`. Esta devuelve un manejador a través del cual accede a los motores.

La librería implementa dos modos de movimiento diferentes con varias funciones: control en posición y control en velocidad. Un control en posición con una función de traslación en línea recta de una distancia concreta (a una velocidad constante) con `VWDriveStraight(manejador,distancia,velocidad)`, giros de un ángulo en concreto (a una velocidad de giro constante) con `VWDriveTurn(manejador,radianes,vel`

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) con `VWDriveCurve(manejador,distancia,radianes,velocidad_angular,velocidad_lineal)`

Estas funciones son no bloqueantes, pueden ser interrumpidas por otras `VWDriveX`. Es decir, la llamada actual a una de esas funciones no terminará si antes se ha llamado a otra función `VWDriveX`. Puedes saber si han terminado mediante la función `VWDriveDone(manejador)` o esperar hasta que termine con la función `VWDriveWait(manejador)`.

El segundo modo de mover los motores consiste en mandar en una sola función la velocidad lineal y la velocidad angular a la que quieres que se desplace el robot mediante la función `VWSetSpeed(manejador,Vlineal,Vangular)`. Esta es una función asíncrona, la consigna se puede variar en cualquier momento.

Para configurar los parámetros internos que se utilizan en los bucles de realimentación con el interfaz VW se usa la función `VWStartControl(manejador,`P_v`,`I_v`,`P_w`,`I_w`)` donde P_v es la constante proporcional para la velocidad lineal, P_w es la contante proporcional para la velocidad angular, I_v el la constante integral de la velocidad lineal y I_w es la constante integral de la velocidad angular. Estos valores son utilizados por el PID de tracción y giro interno de la librería.

Este control puede ser desactivado mediante `VWStopControl(manejador)`. Es recomendable desactivar este control antes de liberar el interfaz de los motores.

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

Estas funciones están descritas con más detalle en las tablas 3.2 y 3.3.

- Servos

  Un servo es un motor que puede ser posicionado y enclavado en un ángulo determinado. Para estos servos no se incluyen encoders y están orientados al control en posición. Nuestro EyeBot dispone de dos servos uno para mover la cámara y otro para el pateador.

  Se utiliza la función `SERVOInit(nombre_servo_en_HDT)` para inicializarlo. Para fijar el servo a un ángulo deseado se necesita la función `SERVOSet(manejador,ángulo)`. Es conveniente liberar el recurso (`SERVORelease(manejador)`) tras finalizar su uso.

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 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.

Cuadro 3.2: Funciones de la interfaz VW: I

3.4. Interacción con el usuario

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

- Pantalla

  Es una pantalla de cristal líquido (LCD, Liquid Crystal Display) que consta de un grid de 128 x 64 píxels, donde se pueden visualizar hasta 17 caracteres ASCII por línea, con un total de 8 líneas.

  Sirve para la comunicación del sistema operativo con el usuario, pues en ella se visualizan mensajes, texto, imágenes... También puede ser utilizada como dispositivo de salida por nuestros programas.

  El programa que se ejecuta en el EyeBot puede escribir en la pantalla cadenas de texto, números enteros, números reales, etc. Por ejemplo `LCDPutInt(entero)`,

Formato Función	*P.Entrada*	*P.Salida*	*Descripción*
Interfaz VW			
int VWStopControl(VWHandle handle)	(handle) Manejador VWDriver.	0 = ok -1= manejador incorrecto	Deshabilita el controlador (PI-Controller).
int VWDriveStraight (VWHandle handle, meter delta, meterpersec v)	(handle) Manejador VWDriver (delta) distancia a conducir en m (positiva adelante) (negativa atrás) (v) velocidad (siempre positiva).	0 = ok -1= manejador incorrecto	Avanza o retrocede el robot el número de metros "deltaçon velocidad "v". Cualquier llamada a VWDriveStraight, -Turn, -Curve o VWSetSpeed interrumpirá la ejecución de este comando.
int VWDriveTurn (VWHandle handle, radians delta, radPerSec w)	(handle) Manejador VWDriver (delta) ángulo de giro en radianes (negativo dirección de las agujas del reloj). (w) velocidad de giro (siempre positiva)	0=ok -1= manejador incorrecto	Hace girar el robot un ángulo "delta"en radianes con una velocidad "w". Cualquier llamada a VWDriveStraight, -Turn, -Curve o VWSetSpeed interrumpirá la ejecución de este comando.
int VWDriveDone (VWHandle handle)	(handle) Manejador VWDriver.	-1= Manejador incorrecto 0 = El vehículo está todavía en movimiento. 1 = El comando previo VWDrivex ha sido completado.	Chequea si el comando previo VWDrivex ha sido completado
int VWDriveWait (VWHandle handle)	(handle) Manejador VWDriver.	-1= Manejador incorrecto. 0 = El comando previo VWDrivex ha sido completado.	Bloquea la llamada a procesos hasta que el comando previo VWDrivex haya sido completado

Cuadro 3.3: Funciones de la interfaz VW: II

que escribe un entero.

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)`). Además se pueden dibujar imágenes a pantalla completa (`LCDPutImage(imagen)`).

El borrado total de la pantalla se realiza con la función `LCDClear()`. Ver tabla 3.4.

- Botonera

  En la parte inmediatamente inferior a la LCD hay una hilera de 4 botones. Se puede acceder a ellos con la función `KEYGet()`, esta función devuelve el botón pulsado y 0 si no ha sido pulsado ninguno.

- Audio

  El micrófono y el altavoz incorporados en el EyeBot posibilitan tanto grabar como reproducir sonidos. También es posible el uso de `AUBeep()` para emitir 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.
LCDPutInt (entero)	Entero que será escrito	Ninguno.	Escribe el número en decimal.
LCDPutFloat (real)	Número que será escrito.	Ninguno.	Escribe el real.
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.
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.
LCDPutImage (img BYTE)	Imagen en blanco y negro (128*64 píxels).	Ninguno.	Imprime la imagen en toda la pantalla.
LCDMenuI (Pos, string)	Posición (1-4). Cadena (4 caracteres máximo)	Ninguno.	Escribe el string en la posición indicada.

Cuadro 3.4: Funciones para la pantalla

3.5. Recursos de multiprogramación

Los sistemas operativos modernos ofrecen la posibilidad de que el usuario pueda desarrollar la programación paralela con varios hilos de ejecución. En este tipo de programación hay distintas hebras, y el microprocesador se reparte entre ellas de modo que todas progresen (pseudo paralelismo) repartiendo el tiempo de CPU para cada proceso en pequeñas cantidades de tiempo para cada hilo, denominadas quantum.

3.5.1. Multihebra cooperativa y con expulsión

Las hebras o hilos (en inglés Threads) son flujos de control independientes dentro de un mismo proceso que comparten datos globales (variables globales, ficheros, etc.), pero poseen una pila y un contador de programa, propios. Se les llama "procesos de

peso ligero" porque su contexto es menor que el contexto de un proceso. Por lo tanto, los cambios de contexto entre hilos son menos costosos que los cambios de contexto entre procesos.

El sistema operativo del EyeBot incluye soporte para programas multihebra. En concreto RoBios ofrece dos modos de multihilo (en nuestro robot), uno es cooperativo y otro con desalojo. En modo cooperativo es el hilo el que ha de ceder explícitamente el flujo para que otra hebras progresen. De hecho si un hilo se bloquea antes de ceder el control el resto se paran. En el modo con desalojo, es el sistema operativo el que se encarga de desalojar de la CPU a la hebra que la esté ocupando, cediendo el flujo de control a otro hilo. Este es el modo con el que hemos trabajado en este proyecto.

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.
OSReady (thread)	Puntero al bloque de control de una tarea.	Ninguno.	Pone el thread a listo.
OSKill (thread)	Puntero al bloque de control de una tarea.	Ninguno.	Elimina el thread pasado y replanifica.
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.
OSPermit	Ninguno.	Ninguno.	Activa el thread pasando a modo PREEMPT.

Cuadro 3.5: Funciones para las tareas en la multiprogramación

El funcionamiento de la multiprogramación en el EyeBot ha de comenzar por una inicialización indicando el método deseado con la función `OSMTInit(modo)`. Tras ello se deberán inicializar las tareas con `OSSpawn(nombre,comienzo,tamaño_pila,prioridad,identifi` Esta función devuelve la tarea 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 con `OSPermit()`. Puedes ver estas funciones en al tabla 3.5.

3.5.2. Semáforos

La implementación de hilos en un sistema operativo convierte a éste en mucho más rápido, pero surgen nuevos problemas de cara al programador, ya que varios hilos de un mismo proceso pueden acceder a una misma variable compartida, lo cual puede ocasionar inconsistencia en su valor. Sin embargo, a veces, la utilización de los hilos simplifica incluso la tarea del programador, dependiendo del tipo de aplicación.

Para evitar que se produzcan estas inconsistencias se suelen proteger las zonas críticas, aquellas zonas a las que acceden varios hilos simultáneamente con semáforos o monitores. En nuestro caso el sistema operativo del robot ofrece semáforos.

Los semáforos se utilizan para coordinar las distintas tareas y garantizar el acceso en exclusión mutua a variables compartidas. Son utilizados en entornos multihilo para proteger variables a las que tienen acceso más de un hilo. Por ejemplo si tienen que acceder un lector y un escritor a un array, lo que lea el lector sea realmente lo que hay en esa variable en ese preciso momento. Supongamos que el lector lee del array y le quitan el tiempo de CPU cuando sólo ha leído la mitad del array, el escritor que tiene ahora la CPU escribe ese array completo de nuevo. Al lector le devuelven el turno y sigue leyendo el array a partir de donde lo dejó produciéndose una condición de carrera ya que la información que tiene el lector no es en realidad la que tiene el array. Puede ser inconsistente.

Cuando un proceso pide un semáforo si no hay nadie ocupándolo lo coge y sigue su ejecución. Si hay alguien esta llamada se queda bloqueada, reteniendo ese flujo hasta que el que había dentro del semáforo lo suelta. Se ponen antes de la zona protegida. Hay que tener cuidado en la programación con semáforos ya que se puede dar el caso que todos lo hilos se queden bloqueados en un semáforo (inanición, bloqueos).

La función `OSSemInit(sem,valor)` lo inicializa. Una vez inicializados pueden ser subidos o bajados con las funciones `OSSemP(sem)` y `OSSemV(sem)` respectivamente. Ver tabla 3.6.

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).

Cuadro 3.6: Funciones para los semáforos

3.6. Compilación y descarga de programas

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 para la descarga de programas.

Cualquier programa que se escriba para el EyeBot se editará en el PC y allí se compila con el compilador cruzado. El fichero *demo.c* se compila con el comando *gcc68 demo.c* proporcionado por dicho compilador cruzado.

El resultado de la compilación es un fichero *demo.hex* apto para ser ejecutado por el microprocesador del EyeBot. Este fichero debe cargase al EyeBot, utilizando un cable serie conectado al ordenador.

El comando dl permite descargar al fichero ejecutable del PC al EyeBot:
dl file.hex
Dentro de la consola del EyeBot se deberá preparar al robot para la recepción de programas antes de ejecutar este comando en el PC.

Capítulo 4

Descripción informática

Una vez descrito el problema a resolver y el robot que vamos a usar, vamos a describir con más detalle la estrategia usada para resolver dicho problema.

Siguiendo la arquitectura JDE, la generación del comportamiento la conseguimos con la ejecución simultánea de 2 esquemas, uno perceptivo y otro motriz. El esquema perceptivo percibe la pelota en la imagen obteniendo su posición. El esquema motriz mueve los motores dependiendo de la posición de la pelota. Hemos implementado este diseño como un programa multihilo donde los dos esquemas descritos se traducen en sendas hebras.

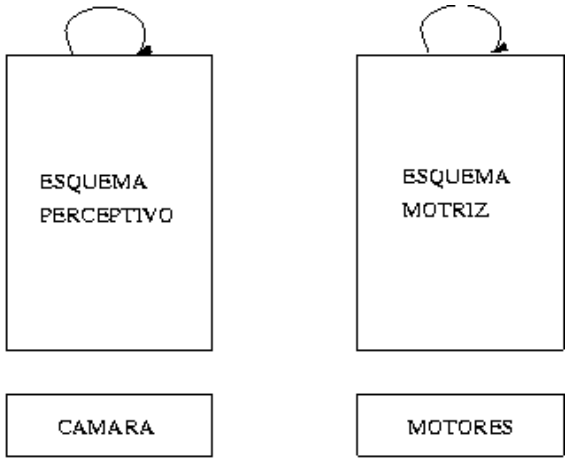

Figura 4.1: Diseño con 2 esquemas

Creamos un hilo de ejecución para cada esquema: un hilo para obtener la imagen y analizarla, y otro para mover los motores. Estos dos hilos se comunican a través de variables compartidas. El hilo perceptor se encarga de actualizar estas variables cada vez que analiza una imagen y el hilo motor las lee para moverse adecuadamente.

Además contamos con otro hilo que chequea periódicamente si se ha pulsado algún botón para principalmente detener la ejecución del programa.

Para seguir la pelota tiene que distinguir ésta en la imagen del resto del entorno mediante un filtro de color. Para el movimiento disponemos de un doble control, en X para el giro y en Y para la tracción.

Además, para mover el robot tenemos varias alternativas, control en velocidad o en posición incremental. En un control en velocidad lo que variamos es la velocidad en función de la distancia a la pelota al centro de la imagen. Éste consiste en mover el robot mediante variaciones en su velocidad, se mueve más rápido cuando la pelota está lejos, despacio cuando está cerca y se para cuando llega a la posición deseada o sus alrededores. Al control en velocidad le pasamos una velocidad lineal dependiente del análisis en Y y una velocidad angular dependiente del análisis en X, el sentido depende del signo del parámetro pasado.

El control en posición consiste en recorrer una distancia o un ángulo predefinidos, cada vez que se analiza una imagen, hasta llegar a la posición deseada. Al control en posición le pasamos la distancia a la que está la pelota, en cada iteración avanzamos y giramos una pequeña distancia que será igual a cero si ya estamos en la posición deseada, en este control también el signo de la distancia indica el sentido de avance y giro. En este control incremental se alcanza el centrado en varias iteraciones.

Para la implementación de ambos controles usamos el interfaz VW proporcionado por el fabricante del EyeBot. Además en el programa se ha tenido en cuenta que el robot ha de seguir la pelota en tiempo real, lo que nos obliga a que los algoritmos usados han de ser eficientes.

En este capítulo vamos a describir detalladamente cómo hemos implementado en el robot el comportamiento sigue-pelota descrito en los objetivos, así como su diseño.

4.1. Descripción general del programa

El diseño del programa principal del proyecto consta de dos esquemas bien diferenciados, uno perceptivo que caracteriza la pelota en la imagen y otro motriz que aprovecha esa información para mover el robot adecuadamente.

Este diseño se materializa en un programa ejecutando tres hebras: una con los algoritmos perceptivos, otra con los algoritmos de control sobre los motores y otra de apoyo, según se observa en la figura 4.2. El hilo de percepción se encarga de la parte

de visión del robot: captura la imagen, la analiza y saca la información relevante sobre la posición de la pelota en la imagen. Busca la pelota en la imagen y extrae su posición y su tamaño. El hilo encargado de los motores es el que, a partir de la posición de la pelota y dependiendo del tipo de control, se moverá de una forma u otra. Este hilo siempre recibe la distancia del punto de la imagen con respecto del centro.

Hay otro hilo de apoyo que sirve para seleccionar con los botones qué cosas se visualizan y para detener el programa. Es principalmente usado para depuración.

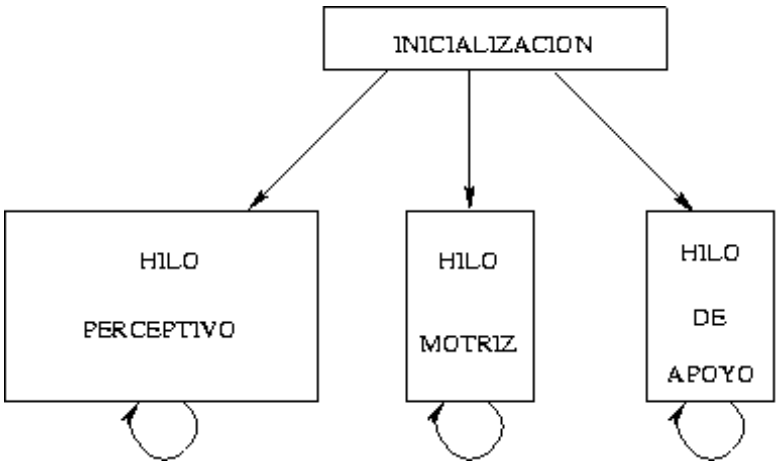

Figura 4.2: Implementación en tres hilos

Cada uno de estos hilos se ejecutan en paralelo de forma independiente, cada uno a su ritmo. El hilo de percepción tiene un ritmo máximo de análisis de 4 imágenes por segundo, el hilo motriz está continuamente ejecutando al ritmo que le permite la CPU, de igual forma que el hilo de apoyo.

En el hilo perceptivo nos interesa buscar en la imagen donde se encuentra la pelota, la cuál es de color naranja. Para localizarla tenemos que distinguir en la imagen lo que es naranja, mediante un filtro de color, agrupar los pixels naranjas en zonas, mediante la segmentación, y, dentro de estas zonas, caracterizar el estímulo pelota.

Para un buen seguimiento es necesario percibir imágenes rápidamente para que el hilo motor lea los datos ya actualizados y el robot se adapte a la nueva realidad mediante la realización del movimiento adecuado. Este control debe ser reactivo para poder movernos en un entorno dinámico (éste lo es ya que la pelota se puede mover). El ritmo máximo de 4 imágenes por segundo viene impuesto por la máxima velocidad de catura(solo para la captura), y al meter algo de procesamiento perceptivo se reduce a entre 2 y 3 imágenes por segundo.

Para mover el robot podemos elegir entre dos controles, en velocidad y en posición. En el control en posición es necesario que los dos hilos estén sincronizados para que el hilo motor no lea varias veces el mismo dato produciéndose el consiguiente error entre la distancia que hay realmente a la pelota y la que ha obtenido el hilo perceptor.

El programa principal es el encargado de activar las distintas hebras, en el modo de multihebra con desalojo anteriormente explicado,e inicializar los semáforos encargados de proteger las zonas críticas, como son las variables compartidas por varias hebras. Primero inicializa los semáforos y luego lanza las hebras con los correspondientes `spawn` para cada una. Este `spawn` devuelve un manejador de hilo que tiene que ponerse a listo antes ejecutar la función `OSReady(premty)` que activa el modo multihilo con desalojo. Activa tres hilos.

4.2. Esquema de percepción

Este esquema esta implementado como un hilo. Primero se encarga de obtener la imagen de la cámara y la analizar para sacar todo aquello que es naranja. Esta operación se realiza aplicando un filtro RGB que sólo se queda con los pixels de color naranja. Este análisis entrega una imagen binarizada.

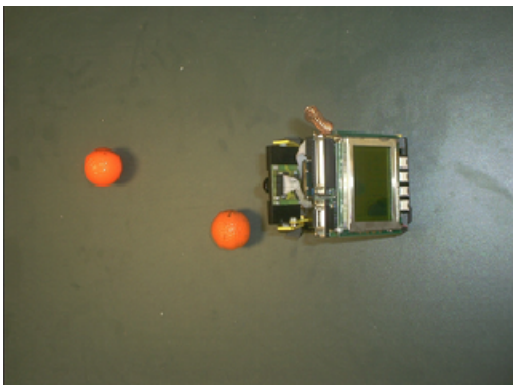

Figura 4.3: Ejemplo varias pelotas

El filtro sólo nos clasifica los pixels pero no nos dice donde está la pelota ni cuantas pelotas se observan en la imagen. Para esto es necesario agrupar los pixels naranjas en segmentos o zonas, partes naranjas independientes unas de otra, partes no conexas sin solapes.

Esta imagen binarizada la volvemos a analizar para agrupar los píxels naranjas en zonas. De estas zonas obtenemos los cuatro puntos que la definen como una ventana

para su posterior análisis. Cada ventana obtenida la asocio con una posible pelota, con esto quiero decir que en cada ventana obtenida puede haber una pelota y como mínimo en una ventana va a haber una.

Una vez calculadas las ventanas busco la pelota dentro de estas. Para obtener la pelota aprovecho que la esfera se proyecta siempre en una circunferencia. Necesito puntos que pertenezcan a esta circunferencia, para ello busco los puntos extremos máximos, esto es, me quedo con el punto naranja más a la derecha de la ventana, el punto más a la izquierda, el que está más abajo y el que está más arriba. A partir de estos puntos puedo sacar el centro de la pelota y su tamaño, aplicando un algoritmo que veremos más detalladamente después.

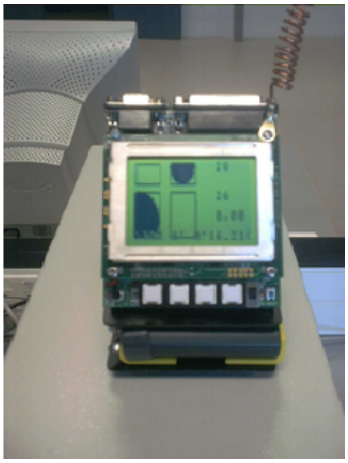

Figura 4.4: Ejemplo de enventanado en el EyeBot

Teniendo todas las pelotas calculadas, tenemos su centro, su tamaño estimado en la imagen y el número de pixels naranja que la componen, debemos decidir que pelota es la que debemos seguir. Nosotros vamos a suponer que la pelota que debemos seguir es la que tiene el mayor número de pixels naranjas. Una vez que tenemos la pelota mayor, calculamos la distancia que hay desde su centro al centro de la imagen y este valor lo almacenamos en variables globales para que puedan ser vistas por el hilo de los motores. Todas las variables que están compartidas por varios hilos están protegidas por semáforos.

La hebra perceptiva funciona en un bucle infinito eficiente que le permite reflejar en las variables de percepción los cambios de posición relativa entre la pelota real y el robot. La velocidad máxima de obtención de imagen es de 4 fps(frames por segundo)y con el procesamiento baja a 2-3 fps, por lo que puede seguir una pelota que no vaya

muy deprisa.

4.2.1. Filtro naranja

Hay varios espacios de color. El filtro le hemos realizado sobre el espacio RGB ya que la imagen obtenida directamente de la cámara trabaja con componentes RGB. Podíamos hacer el filtro en el espacio HSI pero esto implicaba una conversión trigonométrica por cada pixel. Nuestro robot dispone de un emulador para reales lo que hace que vaya más lento al realizar estas operaciones.

```
for (i=0;i<ancho;i++)
  for (j=0;j<largo;j++)
    if ((imagen_col[i][j][rojo]<=255) && (imagen_col[i][j][rojo]>=99) &&
       (imagen_col[i][j][verde]<206) && (imagen_col[i][j][verde]>65) &&
       (imagen_col[i][j][azul]<=184) && (imagen_col[i][j][azul]>=48) &&
       (imagen_col[i][j][rojo]/imagen_col[i][j][verde]<1.8) &&
       (imagen_col[i][j][rojo]/imagen_col[i][j][verde]>1.25))
          imgbin[i*largo+j]=255;
```

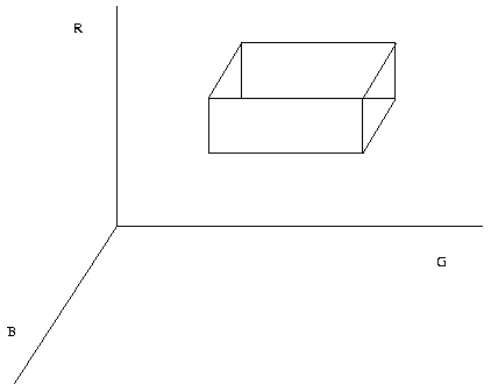

Figura 4.5: Espacio RGB y filtro color naranja

Para solo coger los pixel naranjas primero me quedo con aquellos que están entre unos valores que corresponde con el naranja en RGB. La componente roja ha de estar entre 255-99, la componente verde entre 205-66 y la componente azul entre 184-48. Este margen es demasiado amplio y entraban colores muy parecidos al naranja pero no encajaban con el buscado, además es muy sensible a variaciones en la luz. Para solucionar este problemas introducimos un ratio entre la componente roja y la verde, la división entre la componente roja y la verde a de estar entre 1.8-1.25.

Estos valores han sido ajustados manualmente a nuestra pelota, a partir de las componentes que definen el color naranja en RGB y probados en distintas configuraciones de luz. Este filtro reconoce bastante bien el color naranja incluso con pequeñas variaciones en la luz.

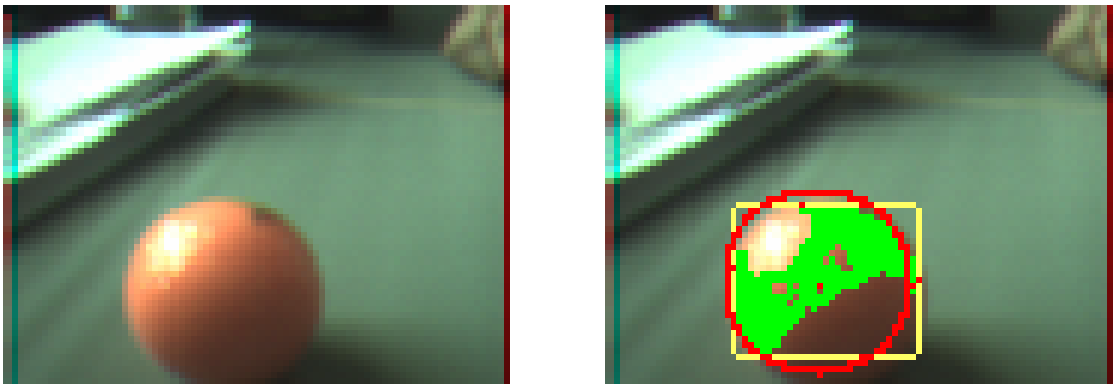

Figura 4.6: Fotos de la pelota y de esta pasado el filtro

Es al aplicar este filtro donde se controla si hay alguna pelota naranja en la imagen, devuelve un valor que representa si hay pelota o no en la imagen. Si el número de pixels en la imagen no supera cierto umbral se estima que no hay pelota en la imagen. En el caso que no haya pelota, no realizamos los demás cálculos con el consiguiente ahorro de tiempo.

4.2.2. Segmentación basada en histograma

Con el filtro hemos identificado los pixels que son naranjas. Sólo con esto no podemos saber donde está la pelota ni cual es su tamaño. Agrupamos los pixels naranjas en zonas, o segmentos, independientes, es decir, que no se solapen unas con otras. El segmentado se podría haber hecho mediante otro algoritmo, como por ejemplo el de calculo de K-medias, que consiste en calcular el numero de pixels naranjas y hacer la media directa. Es computacionalmente costoso, lo que le descarta para nuestra aplicación. El algoritmo de segmentación ha de ser eficiente en tiempo para que el robot pueda seguir la pelota.

Para buscar las distintas zonas naranjas (incluso en el caso de que haya más de una) hacemos un histograma de los puntos naranjas que hay en cada línea, histograma en Y y otro con los puntos que hay en cada columna, histograma en X. Podemos ver estos histogramas en las figuras 4.7 y en la 4.8, respectivamente.

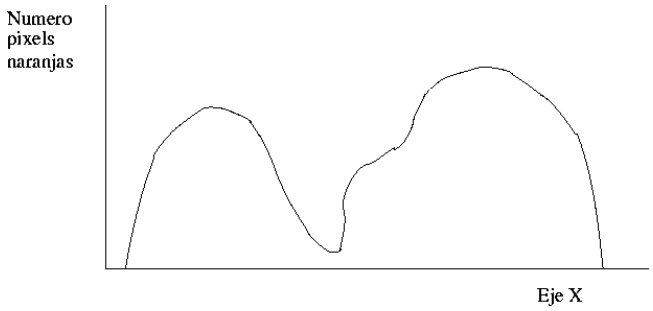

Figura 4.7: Histograma en el eje X

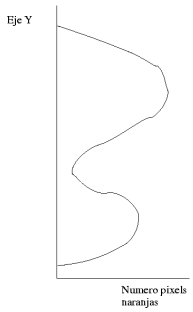

Figura 4.8: Histograma en el eje Y

El objetivo de calcular estos histogramas es hacer un enventanado de la imagen para agrupar las zonas que son naranjas, cada ventana corresponde con una posible pelota. Para calcular estas ventanas buscamos en la imagen zonas con alta densidad de pixels naranjas. Las zonas con puntos naranjas vamos a aproximarlas por ventanas rectangulares en la imagen. De este modo sus bordes verticales y horizontales se pueden buscar por separado en el histograma en X y en Y (respectivamente) bordes de zonas con alta densidad. En concreto los bordes los detectamos cuando el histograma supera cierto umbral de densidad. Ver figura 4.10.

Para obtener los puntos borde de las ventanas a partir de los histogramas probamos dos algoritmos, uno con umbral único y otro con doble umbral.

Con umbral único para obtener el punto de inicio de la zona naranja nos quedamos con aquella línea que supere ese umbral, el punto final de esta zona será el siguiente que se sitúe por debajo del umbral. Se siguen cogiendo puntos de inicio y de fin de zona hasta que se ha recorrido todo el histograma. La desventaja que tiene es que recorta la zona que pertenece a la pelota. Ver figura 4.9.

En el umbral doble el umbral superior se asegura de que hay un número relevante de pixels, que realmente es el comienzo de una zona naranja. El primero expande ese

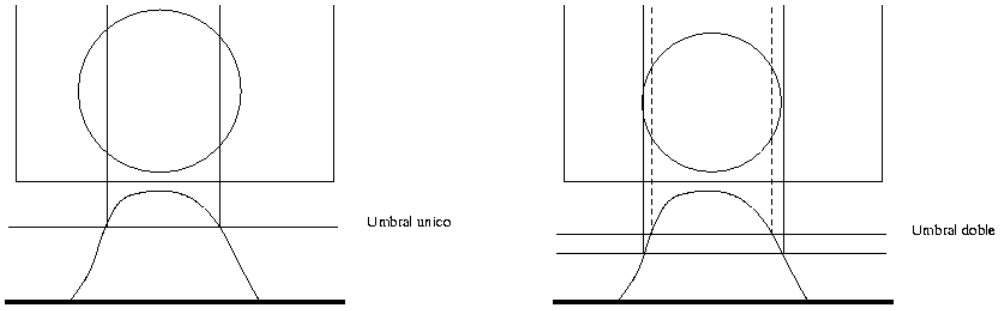

Figura 4.9: Ejemplo de segmentación con umbral único y umbral doble

borde hasta el punto en que se superó este primer borde. Nuestro umbral inferior es de 2 pixels y el superior de 4 pixels. Una vez obtenido el principio de la zona hay que obtener el final, este será la linea que sea inferior al umbral inferior, menor a 2 pixels. Ver ejemplo en la figura 4.9.

Este algoritmo le aplicamos al histograma en X y al histograma en Y obteniendo el principio y el final de las zonas naranjas en los dos ejes.

Una vez que tenemos esta información la combinamos para obtener los 4 puntos que definen cada preventana, son estas preventanas las que después vamos a analizar para reconocer lo que es pelota y lo que no. Combinamos cada par de puntos, principio y final, obtenidos del histograma en X con cada par de puntos obtenidos del histograma en Y.

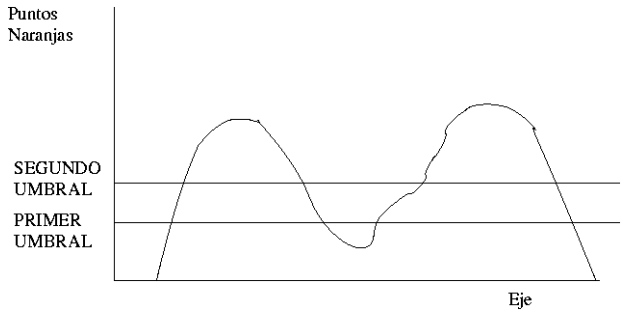

Figura 4.10: Ejemplo de histograma con umbral doble

De estas preventanas sacamos los bordes, para pintarlos en la imagen, que nos delimitan la zona donde debemos caracterizar la pelota. Dentro de estas preventanas puede que no haya ninguna zona naranja, preventana fantasma, por lo que esa preventana es descartada en la fase de análisis. Puedes verlo en la figura 4.11. Un caso real en el robot esta reflejado en la figura 4.4. El enventanado de una sola pelota lo podemos ver

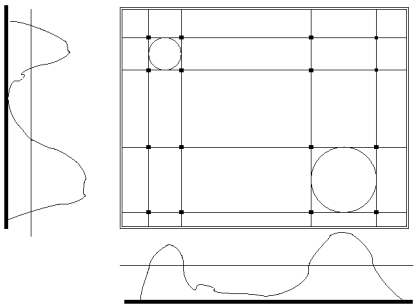

Figura 4.11: Ejemplo de enventanado

en la figura 4.12.

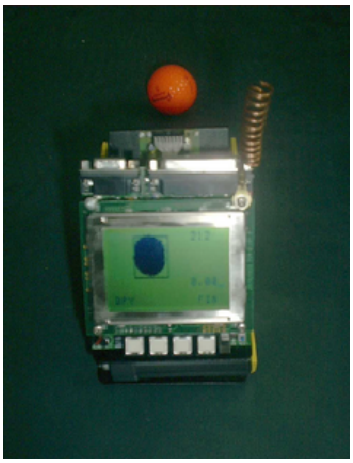

Figura 4.12: Enventanado en el LCD del EyeBot

Necesitamos determinar el centro de la pelota y su tamaño para poder situar la pelota en la imagen y así poder seguirla.

4.2.3. Estímulo pelota

Para definir lo que realmente es la pelota analizamos cada una de estas preventanas. Primero tenemos que recorrer la preventana para definir cuál es el punto naranja que está más a la derecha de la ventana, cual es el que está más a la izquierda, cual está más abajo y cual más arriba.

Este estímulo podemos obtenerlo aprovechando que la pelota real es una esfera y esta se proyecta siempre en una circunferencia. Aplicando que todo punto perteneciente a la circunferencia de un circulo cumple $(X-X_o)^2+(Y-Y_o)^2 = R^2$ podemos obtener el centro, (X_o, Y_o), y el radio, R, resolviendo el sistema de ecuaciones con tres incógnitas y

cuatro ecuaciones(debido a que tenemos 4 puntos), suponiendo que los puntos máximos pertenecen a la circunferencia. Estos puntos extremos muy posiblemente pertenecen a la circunferencia exterior de proyección.

De las 4 ecuaciones obtenidas desechamos siempre una. En un caso normal, los cuatro puntos sean distintos de los máximos de cada ventana, desechamos el punto naranja más a la izquierda. En el caso que algún punto sea igual a un borde de la ventana desechamos ese punto y nos quedamos con los otros tres. Si hay más de un punto que coincide con el máximo, solo desechamos uno.

$(X_1 - X_o)^2 + (Y_1 - Y_o)^2 = R^2$
$(X_2 - X_o)^2 + (Y_2 - Y_o)^2 = R^2$
$(X_3 - X_o)^2 + (Y_3 - Y_o)^2 = R^2$

Despejando las incógnitas nos queda:

$X_o = (-Y_2*{Y_3}^2-Y_2*{X_3}^2+Y_1*{Y_3}^2+Y_1*{X_3}^2-Y_1*{X_2}^2-Y_1*{Y_2}^2+Y_2*{Y_1}^2+Y_2*{X_1}^2+Y_3*{Y_2}^2+Y_3*{X_2}^2-Y_3*{Y_1}^2-Y_3*{X_1}^2)/(2*(X_1*Y_2-X_1*Y_3+X_2*Y_3-X_3*Y_2-Y_1*X_2+Y_1*X_3))$

$Y_o = (-X_2 * {Y_1}^2 - X_2 * {X_1}^2 - X_3 * {Y_2}^2 - X_3 * {X_2}^2 + X_3 * {Y_1}^2 + X_3 * {X_1}^2 - X_1 * {Y_3}^2 - X_1 * {X_3}^2 + X_1 * {X_2}^2 + X_1 * {Y_2}^2 + X_2 * {Y_3}^2 + X_2 * {X_3}^2)/(2 * (X_1 * Y_2 - X_1 * Y_3 + X_2 * Y_3 - X_2 * Y_1 - X_3 * Y_2 + Y_1 * X_3))$

$R = \sqrt{(X_1 - X_o)^2 + (Y_1 - Y_o)^2}$

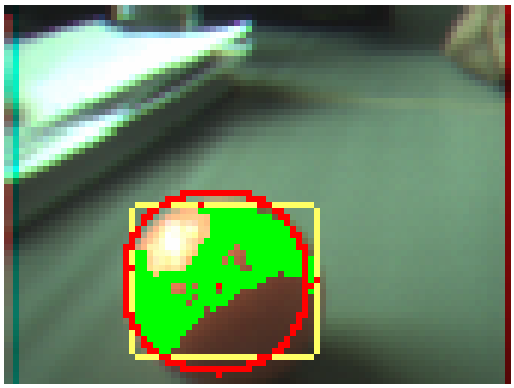

Figura 4.13: Resultado del calculo del estimulo pelota

Este algoritmo es conveniente porque normalmente no toda la circunferencia supera el filtro, sino franjas de la circunferencia (medias lunas). En este caso el centro de la

zona filtrada y el centro de la pelota no coinciden. Un ejemplo es la figura 4.13 obtenida a partir de la figura 4.6.

Este algoritmo es demasiado lento para hacer que el robot se mueva con una cierta vivacidad. Como veremos posteriormente para el control en posición este retardo no se nota considerablemente. Para el control en velocidad lo que necesitamos es analizar el mayor numero de imágenes por segundo y con este algoritmo no lo conseguíamos. Decidimos calcular el estimulo pelota de otra forma aproximada pero que consume menos tiempo de computo.

Nos decidimos por igualar el centro de la pelota con el centro de la ventana calculada. El calculo del centro es mucho más simple. La ventana se define por cuatro puntos, $P1 = (X_{min}, Y_{min})$,$P2 = (X_{max}, Y_{min})$,$P3 = (X_{min}, Y_{max})$ y $P4 = (X_{max}, Y_{max})$. El centro de la ventana se obtendría resolviendo la siguiente ecuación:
$X_o = (X_{max} - X_{min})/2 + X_{min}$
$Y_o = (Y_{max} - Y_{min})/2 + Y_{min}$

Una vez terminada la fase de segmentado y esta de obtención del estimulo pelota ya disponemos de la información de cada pelota en la imagen. De cada una almacenamos en una array su centro, número de pixels naranjas y el tamaño.

4.3. Esquema motriz

El esquema motriz lo implementamos como un hilo. Este hilo se encarga de mover los motores de forma apropiada para que el robot siga la pelota naranja. Hemos desarrollado y comparado dos controles diferentes para realizar esta función, un control en posición y un control en velocidad.

Este hilo lee, de unas variables globales, la distancia de la pelota al centro de la imagen y el tamaño. Con estos datos podemos realizar para el control de giro una realimentación por posición en X y para el control en tracción teníamos dos posibilidades, realimentación por posición en Y o por tamaño de la pelota, pero sólo hemos implementado por posición en Y.

Tenemos delimitada una zona, que equivale al centro de la imagen, donde si la diferencia leída se encuentra en esta zona el robot estará enfrente de la pelota y no se tendrá que mover. Tenemos delimitada la zona tanto en X como en Y. Si la diferencia se sale de este área el robot debe moverse de la forma adecuada. Si el centro de la pelota

está centrado en X pero está alejado en Y, el robot se mueve en tracción para acercarse a la pelota o alejarse si esta demasiado cerca. Si esta centrado en Y pero esta escorado hacia uno de los dos lados, el robot se mueve solo en X hacia el lado correspondiente. Si no está centrado en ninguno de los dos ejes, se mueve en ambos ejes hasta centrar la pelota.

En ambos controles hay que contar con el rozamiento con el suelo, con la inestabilidad y con las oscilaciones.

Para mover el robot hay que inicializar los motores con unos parámetros de control correctos sino se puede dar el caso de que el robot a baja velocidad tiemble.

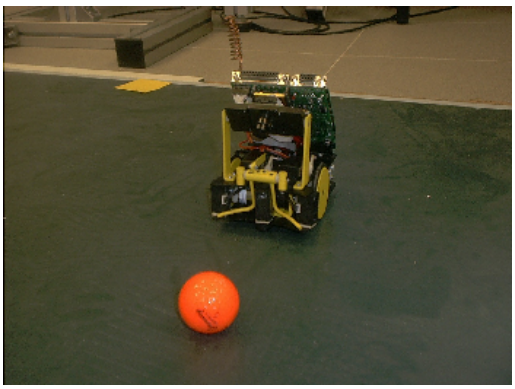

Figura 4.14: EyeBot siguiendo la pelota

4.3.1. Control en posición

El control en posición que hemos desarrollado es un control en posición incremental. El robot por cada iteración avanzará una pequeña distancia y girará un ángulo muy pequeño, si corresponde, según la última imagen analizada. Hemos elegido que sea incremental para darle estabilidad, esto le hace muy estable en el caso de que la pelota este parada. Cada vez que avance o retroceda el robot recorre medio centímetro, y cada vez que gira recorre dos grados. Estos valores han sido calculados empíricamente. Se han elegido esos valores, que no son muy amplios, para que robot no se pase en cada iteración de un extremo al otro y llegue a centrarse con la pelota.

A partir de la distancia recibida sólo tiene que decidir si está dentro de los umbrales en los que se tiene que parar, ya que el sentido del giro viene implícito en el signo de la distancia, si la distancia recibida, según sea la distancia en X o en Y, es negativa girará a la derecha o retrocederá y si es positiva girará a la izquierda o avanzará.

Para el correcto funcionamiento de este control es necesario sincronizar ambos hilos para que cada vez que reciba una imagen realice la operación oportuna. Si el robot lee varias veces la misma distancia sin actualizar, recorrerá esta tantas veces como haya leído el dato sin actualizar produciéndose un error en el desplazamiento. El robot en este caso avanza más de lo que tiene que avanzar.

Hemos usado las funciones `VWDriveStraight(motores,distancia,velocidad)`, para el movimiento en el eje Y, y `VWDriveTurn(motores,ángulo,velocidad)`, para el movimiento en el eje X. Estas funciones también son asíncronas por lo que para asegurarnos que en cada llamada a una de estas funciones termina de ejecutar correctamente usamos la función `VWDriveDone(motores)` en un bucle, hasta que esta función no devuelva 0 como que ha terminado de ejecutar no pasa a la siguiente instrucción. También podemos usar la función `VWDriveWait(motores)` que realiza esa misma operación. Con estas funciones nos aseguramos que realmente en cada iteración recorre la distancia definida anteriormente en Y y gira el ángulo definido anteriormente en X.

4.3.2. Control en velocidad

El control en velocidad consiste en mover el robot comandandole unas velocidades, un velocidad lineal para el avance y una velocidad angular para el giro. Estas velocidades las calculamos a partir de la distancia del centro de la pelota calculada al centro de la imagen. Este esquema no tiene que calcular esta distancia, sólo tiene que leer las variables globales que son actualizadas por el esquema perceptivo.

Para mover el robot es necesario que éste primero supere el rozamiento estático y después se mueva a una velocidad determinada superando el rozamiento dinámico. Para esto hemos dividido el control en dos fases, una donde la velocidad que se le pasa a los motores es mas alta de lo normal para superar el rozamiento estático y moverse, *velocidad de arranque*, y otra donde se le pasa una velocidad calculada con un control proporcional suficiente para superar el rozamiento dinámico, *velocidad nominal*.

Al robot se le pueden comandar 2 velocidades dependiendo del estado en el que se encuentre. Si el robot se esta moviendo directamente le mandamos la velocidad deseada. En el caso de que el robot está parado y quiera moverse se le comandará la velocidad de arranque, o pico de arranque. Si no quiere moverse la velocidad que se le pasará será 0. Ver figura 4.15.

Cuando el robot está parado tiene que avanzar con la velocidad de arranque. Se Comprueba que está parado mediante una consulta de los encoders. Se almacenará el

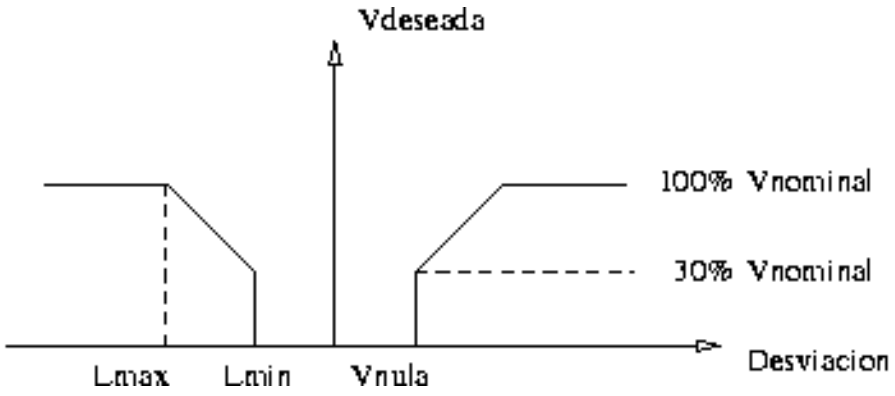

Figura 4.15: Esquema de velocidades en el control en velocidad

valor anterior de estos y se compara con el actual. Si la diferencia no supera un umbral es que el robot está parado por lo que se le comandará la velocidad de arranque. Una vez que ha arrancado en la siguiente iteración, como estamos avanzando, la velocidad que se les pasará a los motores es la velocidad nominal. Esta tiene que superar el rozamiento dinámico para evitar que el robot vaya a tirones. Por ejemplo, si el robot se para y en realidad tiene que avanzar porque no ha llegado a su destino entonces vuelve a saltar la velocidad de arranque produciéndose avances a tirones.

Estas velocidades de arranque y nominales dependen en gran medida de la superficie por la que se va a mover el robot, así como de la carga de la batería. Si la superficie es rugosa la velocidad tiene que ser mayor que en una superficie completamente lisa. Además si la batería esta cargada completamente esta velocidad también deberá ser menor que si está a media capacidad. Hemos obtenido estas velocidades haciendo pruebas en distintas superficies y principalmente en una mesa de ping-pong, que es la superficie de la Robocup.

La velocidad comandada se calcula siguiendo un perfil de control en 3 tramos. En el tramo en que está centrada le comandamos una velocidad nula. En el tramo que superamos un cierto número de pixel la comandamos la velocidad nominal. Para el tramo comprendido entre los dos anteriores aplicamos un control proporcional que consiste en que el robot vaya más rápido cuando la pelota está lejos que cuando está cerca . Queremos que se mueve a la velocidad máxima cuando la distancia sea superior a 30 pixels en X y 20 en Y y que la velocidad mínima sea un 30 % de la velocidad máxima. Esta velocidad mínima se comandará en el punto más cercano a la pelota que se salga de la zona delimitada donde está centrada. Para calcular la velocidad en cada punto aplicamos que la ecuación que define la recta en este tramo es $V = a * D + b$ donde a

y b se definen como:
10030Despejando a y b podemos calcular la velocidad proporcional en cualquier punto. Esto hay que hacerlo para la velocidad lineal, necesaria para la tracción en el eje Y, como para la velocidad angular, necesaria para el giro en el eje X.

Este control proporcional lo usamos para evitar oscilaciones, es decir, que el robot no llegue a centrarse con la pelota. Si al estar cerca va más despacio tiene mas posibilidades de no pasarse y centrarse con la pelota.

Mediante la función `VWSetSpeed(motores,Vlineal,Vangular)`, proporcionada por una librería anexa del fabricante, le comandamos las velocidades a los motores. Esta función es asíncrona, por lo que si se ejecutando y se produce otra llamada a esa función con distintos valores, se ejecuta esta última llamada.

En el control en tracción tanto la velocidad nominal como la de arranque son negativas si el robot tiene que retroceder y positivas si tiene que avanzar. En giro estas velocidades serán negativas si tiene que girar a la derecha y positivas para girar a la izquierda.

Este control en velocidad es más inestable que el control en posición cuando la pelota está estática.

4.4. Hilo de interacción con el usuario

Este hilo se encarga de controlar a los hilos de percepción y motor, además se encarga de interactuar con el usuario. Cuando se pulse cualquier botón del robot es este hilo el que analiza qué botón ha sido pulsado y realiza la acción correspondiente. Cuando nuestro programa está funcionando este hilo muestra un mensaje asociado al significado que le da cada botón según muestra la figura 4.16.

Los botones se empiezan a numerar de izquierda a derecha, el botón 1 es el que está más a la izquierda y el botón 4 es el que está más a la derecha.

El primer botón es usado para mostrar el resultado del análisis en el LCD, por defecto esta opción está activada. Si se pulsa el botón el resultado no se mostrará en el LCD, si se vuelve a pulsar volverá a mostrarse. Con la visualización activada el análisis de imágenes es mas lento pero nos facilita la tarea de depuración (DPY/dpy).

El segundo botón lo utilizamos para cambiar entre control en posición y control en velocidad, por defecto está en control en velocidad. Si pulsamos el botón cambiamos a control en posición. Al cambiar de modo debemos cambiar también los

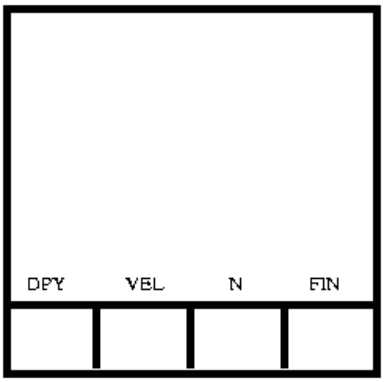

Figura 4.16: Ejemplo de ejecución del programa

parámetros de control que pasamos a los motores cuando inicializamos su control de energía(VEL/POS).

El tercer botón lo usamos para depuración. Por defecto empieza con la opción de movimientos en ambos sentidos, si se pulsa una vez cambiamos a solo movimiento en el eje X, pulsamos otra vez y cambiamos a solo movimiento en Y. Al pulsar una tercera vez volvemos a movernos en ambos ejes. Esta opción es muy útil para depurado de errores(N/A/G).

El cuarto botón termina la ejecución del programa. Al ser pulsado manda las señales de matar a los demás procesos y a si mismo. Antes de enviar esta señal desactiva la cámara y los motores, para dejar la ejecución del programa totalmente acabada y que no se queden activos ninguno de los recursos utilizados(FIN).

Capítulo 5

Conclusiones y mejoras

En este capítulo vamos a ver que objetivos hemos cumplido y posibles mejoras en un futuro de la implementación de este comportamiento.

5.1. Conclusiones

Hemos conseguido generar en el EyeBot el comportamiento sigue-pelota. Para ello hemos realizado un diseño, basado en la arquitectura JDE, con dos esquemas, uno perceptivo y otro motriz. El esquema perceptivo se encarga de obtener la pelota de la imagen y caracterizarla. El esquema motriz se encarga de mover el robot para centrarle con la pelota.

Hemos conseguido implementar este diseño en un porgrama con tres hebras de ejecución. Una hebra para la percepción, otra para el movimiento y otra de apoyo para detener el programa y depurarlo.

En la hebra de percepción hemos desarrollado unas técnicas de filtrado que nos permiten diferenciar la pelota del resto de la imagen. Hemos realizado el filtro en el espacio RGB porque es el que nos ha resultado más eficiente. Otro espacio de color más robusto frente a variaciones en la iluminación es el HSI. Éste le hemos descartado porque la cámara entrega imágenes en RGB y convertirla a HSI es demasiado costoso en tiempo de cómputo. Con este filtro no cumplíamos el requisito de que se ejecutase en tiempo real, requisito que si cumplimos con el filtro en RGB.

También hemos realizado una técnica de segmentación que cumple el requisito de eficiencia, la segmentación por histograma. Este segmentado utiliza ventanas rectangulares para segmentar, además es sencillo de implementar porque no conlleva mucho cálculos. Estos histogramas los calculamos a la vez que pasamos el filtro para ahorrar

pasadas innecesarias a la imagen completa. Hay que recorrer estos histogramas para sacar los bordes de las ventanas, pero lleva menos tiempo que recorrer otra vez la imagen completa.

En este hilo de percepción también hemos realizado un algoritmo para la estimación del centro de la pelota. A partir de tres puntos pertenecientes a la circunferencia de la esfera sacamos su centro y su tamaño.

Para el movimiento del robot hemos implementado dos tipos de control, control en posición incremental y control en velocidad. En el control en posición le comandamos a los motores que avance un incremento de distancia e incremento en ángulo, tanto en giro como en tracción, cada vez que analiza la imagen. Estas distancias son nulas si no hay pelota en la imagen o si la pelota está centrada en la imagen. Le pasamos distancias muy pequeñas para darle vivacidad. Es esto lo que le hace muy estable cuando la pelota esta parada. Debido a la poca velocidad con la que se mueve y que se analizan pocas imágenes por segundo, en este control, si la pelota se mueve el robot no es capaz de seguirla adecuadamente.

En el control en velocidad lo que pasamos a los motores son velocidades. Si la pelota está centrada en la imagen esta velocidad es nula, al igual que si no hay pelota. Para conseguir mover al robot a una velocidad mínima tenemos que superar en un primer momento el rozamiento estático y una vez que está en movimiento superar el rozamiento dinámico. Para superar el rozamiento estático hemos introducido un pico de arranque que solo se va a pasar cuando el robot esté parado, y una vez que se ponga en movimiento le pasamos una velocidad nominal que tiene que superar el rozamiento dinámico.

Este control permite al robot moverse mucho más deprisa que con el control en posición, aunque es más inestable cuando la pelota está parada.

5.1.1. Problemas identificados

Hemos encontrados varios problemas en la realización de este proyecto:

1. El ritmo de percepción de las imágenes es crucial. El ritmo máximo con que el robot captura imágenes es de 4 fps. Con el cálculo de la pelota en la imagen este ritmo se situa entre 2 y 3 fps, 2 con visualización y 3 sin ella. Por este motivo hemos introducido la opción de poder elegir entre visualizar la imagen en el LCD,

útil para depuración, o no. Hemos rechazado también el filtro de color en HSI por el alto tiempo de computo necesario en el robot para realizarlo. Hemos conseguido aumentar este ritmo de percepción realizando de una misma pasada el filtro y el cálculo de los histogramas en X y en Y.

2. Otro problema encontrado es el rozamiento de las ruedas con el suelo. La distinción entre el rozamiento dinámico y el estático ha sido clave. La velocidad con la que teníamos que mover al robot no puede ser muy alta por que provoca excesivas oscilaciones, por el contrario, tampoco puede ser muy baja porque no supera el rozamiento estático. La solución ha sido tener dos velocidades, una de arranque y otra nominal, la que se comanda cuando ya se está moviendo. La velocidad de arranque sólo se da cuando el robot está parado y quiere empezar a moverse.

3. La ejecución del comportamiento es muy sensible a condiciones de iluminación, de carga de las baterías, del material con el que está hecho el suelo. Por ejemplo, con unas condiciones de iluminación distintas a las normales el robot distingue la mano como una posible pelota. Si las pilas están casi agotadas o a media carga al robot le cuesta más moverse que si las tiene totalmente llenas. En un suelo áspero, con mayor rozamiento, encuentra más fuerza de rozamiento y le cuesta más moverse que sobre una superficie lisa y plana.

5.2. Trabajos futuros

Se pueden introducir mejoras en este diseño para conseguir que el análisis de las imágenes sea más rápido. Consiguiendo más imágenes por segundo el comportamiento sería más estable, tanto en el control en posición como en velocidad, pero el gran beneficiado sería el control en velocidad. Por ejemplo, se podría explorar otras técnicas para caracterizar la pelota, como eliminar la segmentación para localizar las zonas naranjas y sustituirla por el cálculo de centro de la pelota como la media de la nube de puntos naranjas. Para tener garantías de que realmente hay pelota el número de pixels naranjas debería superar un umbral mínimo que identificase a la pelota.

Otra posible mejora sería la estimación de la velocidad relativa de la pelota (calculando a partir de la diferencia de la posición en la imagen actual con la posición en la imagen anterior). En la versión actual se corrige el movimiento desde desviaciones de

posición de la pelota medidas en la imagen. Con esta estimación de desviación quizá se podría realimentar mejor el movimiento de corrección a los motores, de modo que se anticipen a la desviación.

Se podría mejorar el filtro haciéndolo más selectivo y adaptables a la iluminación. En este apartado tenemos más problemas ya que es necesario que este filtrado no sea excesivamente caro en computo debido a la poca capacidad que tiene el robot.

Otra mejora es hacer que el robot busque la pelota cuando ésta no se encuentre en la imagen.

Bibliografía

[Cañas01] José María Cañas, Esther García, Vicente Matellán: *Manual del Robot EyeBot.* Informe Técnico GSYC-URJC 2002. Universidad Rey Juan Carlos.

[García01] Esther García Morata: *Construcción de un Teleoperador para el Robot EyeBot* Proyecto Fin de Carrera, Universidad Carlos III, Enero 2002

[Alvarez01] José María Álvarez: *Implementación basada en Lógica Borrosa de Jugadores para la Robocup* Proyecto Fin de Carrera, Universidad Rey Juan Carlos, 2001

[Cañas02] José María Cañas, Vicente Matellán: *Dynamic Schema Hierarchies for an Autonomus Robot* Aceptado como artículo en la VIII Conferencia Iberoamericana en Inteligencia Artificial IBERAMIA 2002 Universidad Rey Juan Carlos, 2002

[Matellan02] Vicente Matellán, José María Cañas: *Interacción persona robot hoy* III Congreso Interacción persona-ordenador, Universidad Carlos III de Madrid, May 2002.

[Aracil97] Antonio Barrientos, Luis Felipe Peña, Carlos Balaguer, Rafael Aracil: *Fundamentos de Robótica* Universidad Politécnica de Madrid, McGrawHill, 1997

[Victor02] Víctor Manuel Gómez: *Comportamiento sigue pared en un robot con visión local.* Proyecto fin de carrera, Universidad Rey Juan Carlos, 2002