INGENIERÍA TÉCNICA EN INFORMÁTICA DE SISTEMAS

Curso académico 2002-2003

Proyecto Fin de Carrera

COMPORTAMIENTO SIGUE PELOTA CON VISIÓN CENITAL

Autora: Marta Martínez de la Casa Puebla
Tutor: Jose María Cañas Plaza

Para mi familia y mis amigos.

Agradecimentos.

Quisiera agradecer al grupo de robótica de la URJC, en particular, a José Mª Cañas por su paciencia y por haberme facilitado la ayuda y los conocimientos necesarios de robótica para la realización de este proyecto.

De igual modo, agradecer también a mi familia y a mis amigos el apoyo ofrecido, sin ellos esto no hubiera sido posible.

Índice general

Resumen **1**

1. Introducción **2**

1.1. Robocup 3

1.2. Grupo de Robótica de la URJC 5

1.3. Comportamiento Sigue Pelota con Visión Cenital 7

2. Objetivos **10**

2.1. Descripcion del Problema 11

2.2. Requisitos 11

3. Herramientas Empleadas **13**

3.1. Video For Linux 13

3.2. EyeBot 18

3.2.1. Actuadores 21

3.2.2. Sistema de Comunicaciones 22

3.2.3. Recursos de Multiprogramación 25

3.2.4. Interacción del usuario con el EyeBot 27

4. Descripción Informática **32**

4.1. Diseño del Comportamiento 32

4.2. Esquema Perceptivo 34

4.2.1. Captura 34

4.2.2. Filtrado 35

4.2.3. Segmentación 37

4.2.4. Identificación de la pelota y del robot 40

4.3. Esquema Motriz 42

4.3.1. Decisiones de Control 43

4.3.2. Transmisión por Radio 44

4.3.3. Eyebot 47

5. Conclusiones y Mejoras **51**

5.1. Conclusiones 51

5.2. Líneas Futuras 53

Bibliografía **54**

Índice de figuras

1.1. Robocup 4
1.2. Diferentes ligas de las que está compuesta la RoboCup 5
1.3. Robots de los que dispone el grupo de Robótica de la URJC. 5
1.4. Robot Pioneer. 7
1.5. Escenario empleado para el comportamiento. 8

2.1. Fotografía del escenario utilizado. 10

3.1. Ubicación del API Video4Linux en el Sistema Operativo LINUX. . . . 14
3.2. Robot Eyebot 18
3.3. Display y botonera 19
3.4. Comunicación entre el PC y el EyeBot. 22

4.1. Comportamiento con dos Esquemas. 33
4.2. Estructura del comportamiento sigue pelota con visión cenital. 33
4.3. Fases para el análisis de la imagen 34
4.4. Imagen capturada por la cámara cenital 35
4.5. Imagen obtenida por la cámara cenital (Izquierda) e imagen filtrada (Derecha). 36
4.6. Formato del color para una configuración de 16 bits 37
4.7. Implementación del color para una configuración de 32 bits 37
4.8. A la izquierda el histograma en el eje X y a la derecha el histograma en el eje Y. 38
4.9. Histograma al que se le ha realizado un Doble Umbral. 39
4.10. Comparativa entre usar un umbral único y uno doble. 39
4.11. Ejemplo del eventanado 40
4.12. Imagen obtenida por la cámara cenital (Izquierda) e imagen segmentada (Derecha). 40
4.13. Ángulos de dos vectores con el eje de abcisas. 42
4.14. Partes del Esquema Motriz 43
4.15. Perfiles de control, Velocidad Angular vs Ángulo (Izquierda) y Velocidad Lineal vs Distancia (Derecha) 44

4.16. Transmisión del mensaje por la radio 46
4.17. Hebras del emovil.c 48

Índice de cuadros

3.1. Funciones del interfaz VW 23
3.2. Funciones del interfaz VW (II) 24
3.3. Funciones para la radiocomunicación 26
3.4. Funciones para las tareas y procesos 28
3.5. Funciones para la pantalla 30
3.6. Funciones para la pantalla (II) 31
3.7. Funciones para el teclado 31

4.1. Rango de valores HS para la pelota y el EyeBot 36

Resumen

El presente proyecto consigue que un robot sea capaz de seguir a una pelota mediante el uso de una cámara cenital como sensor principal. En concreto usamos el mismo escenario que se emplea en la RoboCup, ya que este comportamiento está realizado con vistas a participar en un futuro en ella. En este escenario hay una cámara situada sobre el terreno de juego que continuamente va capturando imágenes. El terreno de juego es un tablero de ping pong y sobre él van a estar situados una pelota de golf naranja, y un robot EyeBot. Hemos usado la plataforma GNU/Linux para la realización de este proyecto y el lenguaje utilizado ha sido C.

Dentro de este proyecto se mezclan varios temas relacionados con la informática, como la visión computacional, la robótica y las comunicaciones. Para capturar la imagen con la cámara cenital, nos hemos apoyado en la interfaz Video4Linux, recientemente estandarizada en Linux para la digitalización de imágenes. La cámara está conectada al PC, y éste será el encargado de analizar la imagen mediante técnicas de visión computacional, como el filtrado de color y la segmentación, con la finalidad de hallar el ángulo y la distancia relativas entre la pelota y el EyeBot.

Con estos datos de la imagen, tomamos las decisiones de control adecuadas: si la pelota está lejos el robot deberá ir a más velocidad, o si está cerca a menos, si está a un lado girará hacia ese lado, ... Al tomarse estas decisiones en el PC, deberemos enviar un mensaje al EyeBot estableciendo para ello, una comunicación por radio. En el EyeBot se ejecuta un programa encargado de recibir este mensaje y de materializar en los motores las órdenes de movimiento recibidas.

Capítulo 1

Introducción

La palabra robot deriva de la palabra checoslovaca *robota*, que significa trabajador, sirviente. Surge en la obra RUR, los *"Robots Universales de Rossum"*, escrita por Karel Capek. En esta obra se plantea la construcción de robots para liberar a las personas de la carga pesada del trabajo. A su vez, el escritor Isaac Asimov, a quién se atribuye el término robótica, previó un mundo futuro en el que existían reglas de seguridad para que los robots no pudieran ser dañinos para los seres humanos. En contraste con estas dos ficciones, en 1954, George Devol desarrolló un brazo primitivo o manipulador que sería el origen real de los robots industriales.

En poco tiempo, la robótica fue pasando a ser una realidad imprescindible en el mundo industrial, ya que da lugar a procesos de producción mucho más eficientes y a una mayor calidad de los productos. Los robots industriales están programados para realizar sólo determinadas tareas repetitivas, como por ejemplo realizar soldaduras en una cadena de montaje para el automóvil. A finales de los años 70 y a principios de los 80, surgió una nueva aproximación en la robótica en el mundo de la investigación. Este nuevo tipo de robótica se denomina *Robótica Autónoma*. Los robots autónomos son sistemas completos que operan eficientemente en entornos complejos sin necesidad de estar guiados y controlados por operadores humanos. Una propiedad fundamental que poseen es la de poder reaccionar adecuadamente ante situaciones no previstas explícitamente en su programación previa. Por ello, se considera a la robótica autónoma un campo muy relacionado con la inteligencia artificial, puesto que para generar un comportamiento adecuado se requiere una cierta inteligencia mínima.

Históricamente, los robots han sido utilizados en distintas tareas. Muchas de las que realizan son aquellas peligrosas o desagradables para los seres humanos, como exploraciones submarinas o volcánicas. También, se han utilizado robots para exploraciones espaciales, como la realizada por la sonda espacial Galileo de la NASA, que viajó a Júpiter en 1966 y realizó tareas como la detección del contenido químico de la atmósfera de dicho planeta. De igual modo, se han usado robots dentro del campo de la

medicina, sobre todo asistiendo en operaciones quirújicas. Dentro del entretenimiento, se han desarrollado robots para hacer de animal de compañía, como el perrito Aibo de Sony [1].

1.1. Robocup

La RoboCup [2] es un ambicioso proyecto cuya idea es formentar la investigación en los campos de la Robótica y de la Inteligencia Artificial (IA), resolviendo para ello un problema estándar. Este problema consiste en el juego del fútbol. Se ha elegido este juego porque supone un escenario competitivo en donde se dan problemas típicos de la Inteligencia Artificial y la robótica, tales como generar comportamientos (chutar gol, pasar una pelota, ...), localizar jugadores, ... Además, es un escenario dinámico y vivaz, pues la situación cambia rápidamente, lo que obliga a tomar decisiones en tiempo real. No nos olvidemos, además, del gran aliciente que provoca el juego, lo que implica la facilidad de hacer llegar la iniciativa al público. El ideal de la RoboCup es conseguir que un equipo formado por robots sea capaz de ganar al mejor equipo de fútbol del mundo formado por humanos en el 2050. Cabe mencionar, que recientemente se ha planteado otro escenario, la RoboCupRescue, en la que un grupo de diversos agentes con diferentes características interactúan con un fin específico, salvar vidas en situaciones de desastre, como en incendios, bloqueos de vías, derrumbamientos de construcciones, ...

Alan Macworth, en 1992, fue el primero en sugerir la idea de que los robots jugaran al fútbol. Independientemente, esta idea también fue examinada por un grupo de investigadores japoneses. Tenían la idea de promocionar la ciencia y la tecnología a través de este juego. Tras ver que el proyecto era interesante, se creó en Japón la Robot J-League, que al poco tiempo, despues de ver el interés mostrado por otros investigadores fuera de Japón, acabó denominándose Robot World Cup Iniciative. El encargado de llevar a cabo todo esto, fue el japonés Hiroaki Kitano, empleado de la empresa Sony.

A finales de agosto de 1997 se realizó la primera competición oficial de la RoboCup en Nagoya. Previamente, en 1996, se organizó una Pre-RoboCup para poder analizar los posibles problemas de una competición de este tipo a nivel mundial. Asociada a la primera Competición Mundial de Fútbol para Robots y sus Conferencias, tuvo lugar el primer Taller Internacional de la RoboCup, cuya finalidad era permitir un intercambio de información que facilitase la investigación.

Dentro de la competición de la Robocup, existe una serie de modalidades. Cada

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

Figura 1.1: Robocup

una de ellas, tiene su propia regulación en cuanto a número de robots, el tamaño de estos, las dimensiones del campo, ... Actualmente las modalidades existentes son:

- Liga de Simulación: Los jugadores son agentes de software sobre un servidor que simula las situaciones de juego con base en el comportamiento de los robots de cada equipo y la pelota. También, este servidor hace las veces de árbitro. (Esquina superior izquierda de la Figura 1.2)

- Liga de Robots Pequeños: Cada equipo consta de cinco robots, los cuáles se comunican entre sí y con un ordenador principal a través de un sistema de radio. Una cámara situada sobre el campo de juego permite al ordenador central estimar la situación de los jugadores y de la pelota. Éste planifica y ordena las acciones de los robots, los cuáles pueden llegar a tener cierto grado de autonomía. El terreno de juego es un tablero de ping pong de color verde, y el balón utilizado es una pelota de golf naranja. Cada robot cuenta con etiquetas identificativas de colores y tiene un tamaño limitado por la organización: deberá entrar en un cilindro de 180 mm de diámetro y 150 mm de altura (Esquina superior derecha de la Figura 1.2).

- Liga de Robots Medianos: Consta de 5 robots por equipo. Cada robot dispone de una cámara local y todo el procesamiento de su comportamiento se realiza localmente en el ordenador de a bordo (Esquina inferior izquierda de la Figura 1.2).

- Liga de Cuadrúpedos: Está patrocinada por Sony y los equipos estarán formados por robots aibos, los cuáles tiene forma de perro (Esquina inferior derecha de la Figura 1.2).

- Liga de Humanoides: Surgió en el 2002, pero en la actualidad es sólo de demostración.

Figura 1.2: Diferentes ligas de las que está compuesta la RoboCup

1.2. Grupo de Robótica de la URJC

El grupo de robótica de la URJC [3] es un grupo formado por alumnos y profesores de esta universidad, que trabajan con temas relacionados con la robótica y la Inteligencia Artificial aplicada (IA). Surgió en el año 2000, y dispone de 30 robots Lego, 6 robots Eyebot, 1 robot Pioneer y un perrito Aibo de Sony (figura 1.3).

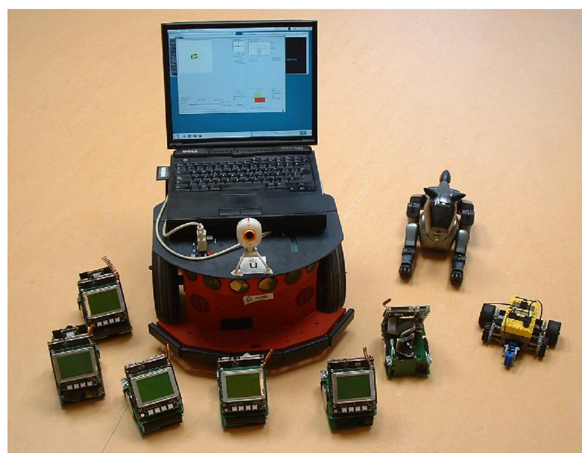

Figura 1.3: Robots de los que dispone el grupo de Robótica de la URJC.

[3] http://gsyc.escet.urjc.es/robotica

Existen dos líneas de trabajo principales en el grupo, por un lado está la creación de un equipo de fútbol para competir en la anteriormente citada RoboCup, y la generación de comportamientos autónomos en entornos de interiores, como laboratorios u oficinas.

Inicialmente, el grupo se planteó la creación de un equipo de fútbol capaz de competir tanto en la liga simulada como en la categoría de pequeños robots dentro de la RoboCup. Para participar en la categoría de robots pequeños, previamente hubo que elegir el robot que se iba a emplear. Éste no debería superar los 180 mm de largo y ancho; y los 150 mm de altura. Después de descartar los robots Lego, se decidió comprar, tras varias comparativas entre robots similares, el robot EyeBot [4] (descrito en el tema 3), ya que económicamente era asequible, poseía una cámara local y tenía una capacidad de proceso buena a la hora de tomar decisiones y procesar datos.

Para conocer de qué era capaz, y de qué recursos disponía el robot EyeBot, se realizó un proyecto consistente en la elaboración de un teleoperador [García02] para él. Éste permitía el movimiento del robot y capturar datos sensoriales adicionales. También, se diseñaron proyectos para establecer comunicaciones entre distintos EyeBot, como el realizado por Carlos Agüero, que consistía en un protocolo de encaminamiento para Redes Adhoc [Aguero02]; y el de Ana Martínez, que elaboró una biblioteca fiable para la comunicación inalámbrica entre los Eyebots [Martinez03].

También, se diseñó la implementación basada en lógica borrosa para jugadores de la RoboCup [Alvarez02], para la participación en la liga simulada. Continuando con la línea de la RoboCup, el presente proyecto estaría implementado para la participación en la liga de pequeños robots.

Otra de las líneas del grupo es la generación de comportamientos autónomos en entornos de oficina. Para este tipo de comportamientos, se diseñó la arquitectura JDE (Jerarquía Dinámica en Esquemas) [Cañas02]. Ésta es una arquitectura de control basada en esquemas. Hay dos tipos de esquemas, los perceptivos y los motores. Los perceptivos generan una información para que sea utilizada por otros esquemas, a partir de unos datos sensoriales. Los esquemas motores generan las respuestas del robot basándose en la información dada por los esquemas perceptivos. Estos esquemas (tanto los perceptivos, como los motores), se organizan en un árbol jerárquico. De esta forma se pueden ir creando comportamientos cada vez más complejos a partir de esquemas más básicos. Un ejemplo de comportamiento autónomo en entorno de oficina que sigue la arquitectura JDE es el sigue pared con visión local [Gomez02].

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

Uno de los robots de los que disponemos para este tipo de comportamientos en interiores es el robot Pioneer (figura 1.4). Es un robot de tamaño mediano, sensorialmente dotado de una corona de 16 ultrasonidos, un cinturón de sensores táctiles y un encóder en cada rueda. Tiene tres ruedas: dos motrices, con sendos motores, y una rueda loca. Es una plataforma genérica muy difundida en la comunidad robótica, que permite aplicaciones como construcción de mapas, navegación, etc. El cómputo necesario se realiza en un microcontrolador interno y un ordenador portátil que se coloca encima de la plataforma móvil. Ambos se comunican a través de puerto serie. Adicionalmente le hemos añadido una cámara en color conectada al portátil por el puerto USB. Este robot ha sido empleado en proyectos como el de evitación de obstáculos basado en ventana dinámica [Lobato03] y en el sigue persona con visión, este último todavía en curso.

Figura 1.4: Robot Pioneer.

1.3. Comportamiento Sigue Pelota con Visión Cenital

Siguiendo la línea de investigación del grupo el presente proyecto está enfocado a una posible participación en la RoboCup. Su objetivo principal es conseguir seguir a una pelota, en tiempo real, mediante las imágenes capturadas por una cámara cenital. Para ello usaremos el mismo escenario usado en la ya citada RoboCup (Figura 1.5), tendremos una cámara situada sobre el terreno de juego que va capturando continuamente imágenes. Estas imágenes serán analizadas por un PC con la finalidad de poder hallar en ellas las posiciones de la pelota y del robot. Si la pelota se mueve el EyeBot tendrá que ir tras ella, hasta que o bien la pelota se pare y el robot se detenga frente a ella, o bien la pelota haya desparecido. Como el análisis de la imagen y la posterior toma de decisión de control se toma en el PC, esta información deberá ser

enviada desde el PC al EyeBot.

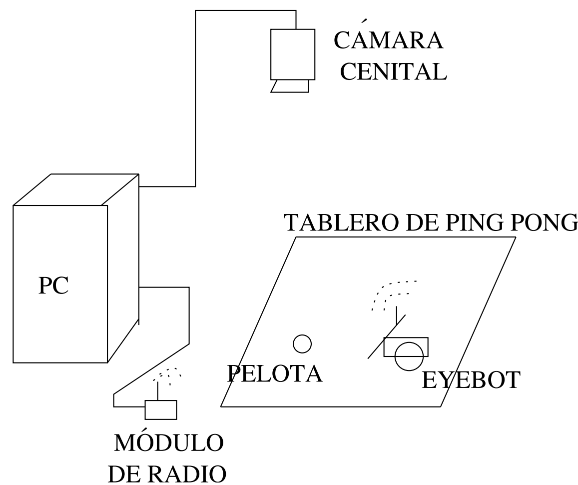

Figura 1.5: Escenario empleado para el comportamiento.

El precursor principal para la realización de este proyecto fue el comportamiento sigue pelota con visión local [SanMartín02], proyecto realizado por Félix SanMartín. La principal diferencia entre el nuestro y el anterior es que en nuestro caso usamos una cámara cenital conectada al PC como sensor principal y no una cámara local en el robot. En este sentido tenemos dos plataformas de cómputo: el propio robot y el PC. Por ello hemos implementado dos programas, uno de ellos se ejecuta en el PC, y analiza las imágenes para hallar las posiciones de la pelota y del robot. Y el otro corre en el EyeBot, encargado de mover los motores de acuerdo a los datos obtenidos en el PC. En el proyecto sigue pelota con visión local, el sensor principal es la cámara que posee el EyeBot y se ejecuta un único programa en el robot, encargado de realizar todos los cálculos. Una ventaja de haber realizado los cálculos en el PC, es que éste posee una mayor capacidad de proceso que el EyeBot, por lo que obtenemos unos resultados más rápidos. Esto conlleva potencialmente una mayor calidad de movimiento que el que se daba en el proyecto sigue pelota con visión local.

Esta memoria consta de cinco capítulos. En el segundo capítulo describiremos cuál es el objetivo concreto del comportamiento y qué condiciones se han de cumplir para un correcto funcionamiento del mismo. En el tercero describiremos las herramientas necesarias para su implementación, como son video4linux para la captura de la imagen y la interfaz de programación del robot EyeBot. En el capítulo cuatro explicaremos cómo hemos desarrollado el comportamiento, primeramente comentaremos su diseño conceptual para posteriormente pasar a detallar los programas que lo implementan. Finalmente, presentamos en el capítulo 5 las conclusiones que hemos obtenido (qué objetivos hemos realizado y con qué problemas nos hemos encontrado), y qué mejoras se

pueden efectúar para posibles trabajos futuros.

Capítulo 2

Objetivos

Una vez situado el contexto de este proyecto, vamos a describir cuál es su objetivo, y como se divide en varios subobjetivos. También, describiremos el escenario en donde se efectuará el comportamiento. En concreto, usaremos el mismo escenario que se emplea en la Robocup, en el que se tiene una cámara cenital situada en el techo, sobre un tablero de ping pong que hace las veces de terreno de juego, en donde estarán situados los robots EyeBots y una pelota de golf naranja (Figura 2.1)..

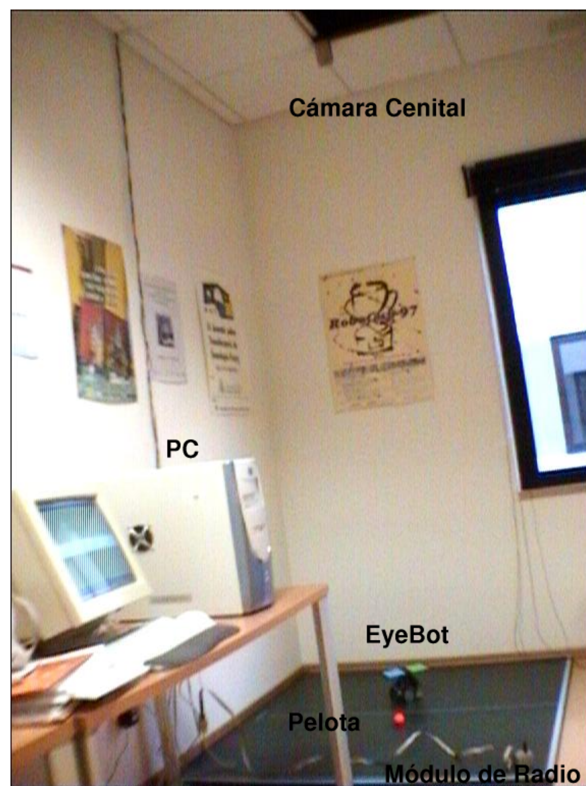

Figura 2.1: Fotografía del escenario utilizado.

2.1. Descripcion del Problema

El objetivo principal de este proyecto es que el EyeBot siga a la pelota en tiempo real manteniéndose a una cierta distancia, es decir, si la pelota se mueve hacia un lado, el robot deberá girar hacia ese lado, si la pelota se mueve hacia delante el EyeBot deberá hacer lo mismo, etc . Para ello, hemos dividido este objetivo en tres subobjetivos. El primero sería resolver el tema de la percepción, en donde una cámara situada sobre el terreno de juego (en donde está situada la pelota que se va moviendo, y el robot), va capturando continuamente imágenes. Estas imágenes serán analizadas por un ordenador mediante técnicas de filtrado y segmentación, para poder hallar la posición en la que se encuentra la pelota y el EyeBot. Si queremos que el robot siga a la pelota en tiempo real, tendremos que analizar estas imágenes en el mínimo tiempo posible.

El segundo subobjetivo sería desarrollar un sistema de control que permita tomar las decisiones de movimiento adecuadas. Para ello, hay que determinar los comandos motores concretos que materializan un buen seguimiento de la pelota en las distintas situaciones. Esto es, si la distancia entre el EyeBot y la pelota es grande, el robot deberá ir a más velocidad, en contra, si la distancia fuera pequeña, la velocidad sería menor. Al igual que si la pelota está situada a un lado del EyeBot, le diríamos que girara hacia ese lado. El EyeBot deberá pararse cuando la pelota ya no se encuentre en movimiento y se encuentre situado frente a ella. Tampoco deberá moverse cuando la pelota no se encuentre en el terreno de juego, es decir, cuando por algún motivo se haya extraviado.

Ya que el sistema de control se implementa en el PC, para que el robot realmente se mueva y pueda seguir a la pelota, esos comandos han de llegar hasta él. Para ello, disponemos de una comunicación por radio entre PC y EyeBot, pero la biblioteca que nos proporciona el fabricante no es fiable. Por esto, nuestro último subobjetivo es la implementación de un sistema de transmisión robusta entre el PC y el EyeBot.

2.2. Requisitos

Para la realización de este proyecto contamos con unos requisitos de partida que condicionan su desarrollo. Éstos se han de cumplir siempre para que nuestro comportamiento se ejecute de una manera eficiente y correcta.

En primer lugar, la edición de los programas necesarios para este proyecto se hará en el lenguaje de programación ansi-C, puesto que la interfaz de programación del Eye-

Bot está implementada en ese lenguaje. Además, utilizaremos un sistema operativo GNU/Linux, que es el que utilizamos en el laboratorio de robótica. El código resultante, será software libre, por lo que se podrá utilizar y modificar de acuerdo con la licencia GPL (General Public License) publicada por la Fundación de software libre.

En segundo lugar, deberemos Diseñar el comportamiento siguiendo la división esbozada en el apartado anterior y de acuerdo a la arquitectura JDE [Cañas02]. Utilizaremos esta arquitectura porque es la propuesta por el grupo de robótica. Siguiendo esta arquitectura, dividiremos al comportamiento en pequeños esquemas.

La tercera condición es que, a la hora de generar el comportamiento sigue pelota con vision cenital, este ha de ser lo más robusto posible, es decir, que funcione correctamente en distintos entornos con diferentes condiciones físicas, como cambios en la luminosidad (que funcione con luz natural, artificial, etc), y que no haya que cambiar nada de su implementación. Esto es importante, porque si nuestro equipo de la Robocup va a un campo con condiciones físicas diferentes al nuestro, el comportamiento deberá seguir funcionando correctamente.

Otro requisito indispensable es que el análisis de imágenes sea en tiempo real, ya que como la pelota se va moviendo continuamente, y el EyeBot tiene que ser capaz de seguirla, se deben de analizar las imágenes en el mínimo tiempo posible, porque sino, no habría impresión de vivacidad y no iría a tiempo real. Como referencia hemos de ser capaces de analizar las imágenes a un ritmo superior de 4 imágenes por segundo, para que el control sea vivaz. A un ritmo menor no se conseguiría el propósito deseado. Una de las ventajas que tenemos frente a otros comportamientos, como el del sigue pared [Gomez02] y el sigue pelota con visión local [SanMartín02], a la hora de conseguir un procesado rápido de imágenes, es que el análisis de éstas se realiza en el PC, que tiene un procesador más potente que el EyeBot, por lo que previsiblemente permite tratar un gran número de imágenes en poco tiempo. Esto facilitará que haya más calidad de movimiento frente a los citados comportamientos.

Capítulo 3

Herramientas Empleadas

Antes de abordar la materialización concreta del comportamiento, en este capítulo vamos a describir dos herramientas fundamentales en las que nos hemos apoyado para su implementación. Por un lado tenemos la interfaz video4linux para poder capturar las imágenes de la cámara cenital. Y por el otro la interfaz de programación del robot EyeBot, mediante la cuál podremos acceder a los distintos recursos de los que dispone el robot. A continuación, pasamos a describirlas.

3.1. Video For Linux

La visión se puede contemplar como un sensor más que los robots pueden utilizar para inferir el estado de su entorno. Es un sensor muy complejo que entrega un flujo contínuo y abundante de imágenes. En el grupo de robótica de la URJC tenemos varios escenarios típicos donde utilizamos las imágenes capturadas por una cámara. Podemos usar la cámara local del robot, como en el caso del sigue pared [Gomez02] y el sigue pelota con visión local [SanMartín02]. Y también, utilizar una cámara cenital, como la que vamos a emplear para nuestro comportamiento. Esta cámara estará situada sobre el terreno de juego y la utilizaremos para hallar la posición del robot y de la pelota. En concreto empleamos una cámara `CCD de SUN de vídeoconferencia` conectada a una tarjeta capturadora `BT878`.

Como para este proyecto hemos utilizado el sistema operativo Linux, emplearemos para capturar imágenes la intefaz video4linux. Esta interfaz homogeniza el acceso a las imágenes con independencia de la cámara concreta que se trate, y da soporte a diferentes capturadoras de video y televisión, como también a cámaras USB, y ha sido estandarizada recientemente para este sistema operativo.

Para cada dispositivo que tengamos conectado, video4linux asigna una serie de entradas en el directorio `/dev`: los dispositivos de vídeo tienen asignados la entrada `/dev/videoXX`, y son los que permiten la captura de señales de vídeo. Ejemplos de

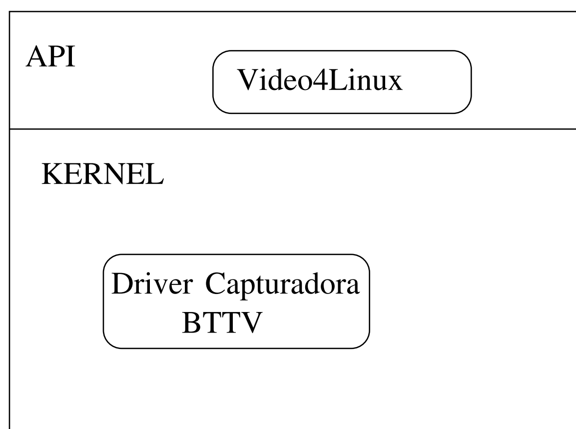

Figura 3.1: Ubicación del API Video4Linux en el Sistema Operativo LINUX.

estos dispositivos son los sintonizadores de televisión o las cámaras. Los de audio, tienen la entrada en `/dev/radioXX` y permiten el manejo de dispositivos "sólo audio", como las tarjetas de radio, descompresores mp3, sonido de videoconferencias, etc. Los dispositivos de información, como sistemas de videotexto y teletexto, tendrán la entrada en `/dev/vtxXX`. Por último, los dispositivos de WebCast e InterCast, que son una nueva tecnología de distribución de páginas web a través de la señal de televisión, tienen su entrada en `/dev/vbiXX`.

Desde el punto de vista de sus componentes, VideoForLinux consta del módulo `videodev` como núcleo del sistema de vídeo. Depende de diversos módulos auxiliares según el hardware concreto que tengamos instalado, como: `i2c` e `i2c_chardev` son dos módulos genéricos que sirven para gestionar el bus `I2C`. El segundo de estos módulos permite el manejo en modo carácter de un bus `i2c` (si existe el dispositivo `/dev/i2cXX`). `bttv`, es el módulo de control de las tarjetas de vídeo basadas en el chip `Bt848`. `msp3400`, controla el chip `msp34XX` que es un decodificador estéreo incluido en muchas tarjetas de vídeo. Y el módulo `tuner` que permite el manejo de dispositivos de vídeo o audio sintonizables, proporcionando datos que el módulo `bttv` puede manejar. Por ejemplo, en nuestro caso, para la capturadora `BT878` usaremos el módulo `bttv`. Además de los descritos, cada dispositivo particular dispone de su propio módulo. Estos módulos interaccionan con su dispositivo, propocionándole el acceso a cada tarjeta.

Si queremos que funcione el kernel del sistema debemos incluir el soporte específico para él. Para ello, podemos recompilar un kernel que los incluya dentro, o mejor, incluirlos en el kernel como módulo.

Las aplicaciones interactúan con el módulo `videodev`. Éste les proporciona información sobre los diversos dispositivos, y, además, permite una serie de operaciones.

Como podemos ver a continuación, para capturar imágenes utilizando video4linux hemos hecho uso de esta secuencia:

- Apertura/cierre de dispositivos de video
- Identificación de las prestaciones de cada dispositivo, por ejemplo si la tarjeta tiene sintonizador de televisión, o tiene varias entradas de señal de vídeo.
- Seleccionar el modo de funcionamiento.
- Lectura de imágenes

Primero abriremos el dispositivo con la llamada al sistema `open`, en donde le enviaremos como parámetro el nombre que el sistema asigna a dicho dispositivo: `/dev/video`.

```
/** Apertura e inicializacion del dispositivo de video4linux **/
 fd = open(video, O_RDWR);
 if (fd < 0)
   {
     perror(video);
     exit(-1);
   }
```

Después averiguaremos las características que posee el dispositivo de video. Para ello utilizaremos la función `ioctl`. En la llamada `ioctl(fd, VIDIOGCAP, &cap)`, el parámetro `VIDIOGCAP` nos rellena una estructura del tipo `video_capability` que nos indica, el nombre de la interfaz, sus capacidades, el número de canales de vídeo y de audio, y, el tamaño máximo y mínimo en píxeles de la imagen que puede capturar. Con `ioctl(fd, VIDIOCGCHAN, &chan)` se nos devolverá información referente al número del canal, su entrada asociada, el número de sintonizadores que posee y sus tipos. Si dispone de señal de audio y el tipo de entrada de vídeo. Utilizando `ioctl(fd, VIDIOCSCHAN, &chan)` se nos permitirá seleccionar el canal deseado sobre el dispositivo escogido. Para conocer las características del sintonizador, disponemos del parámetro `VIDIOCGTUNER` Y `VIDIOCSTUNER`. `VIDIOCGTUNER` nos dará el número de sintonizadores, su nombre (AM,FM,TV), el rango de frecuencias, el tipo de señal (pal, ntsc, secam,....), la calidad de la señal, y el tipo de señal de audio (mono, estéreo). Y con `VIDIOCSTUNER`, activaremos el sintonizador deseado sobre el canal activo en cada momento.

```
if (ioctl(fd, VIDIOCGCAP, &cap) < 0){
 perror("VIDIOGCAP");
 fprintf(f1, "(" video " not a video4linux device?)\n");
 close(fd);
```

```
 exit(1);
}
for(i=0; i<cap.channels; i++){
 chan.channel=(int) i;
 if (ioctl(fd, VIDIOCGCHAN, &chan) == -1){
  perror("VIDIOCGCHAN");
  close(fd);
  exit(1);
 }
}

if ((strcmp(cap.name,"BT878(Hauppauge new)")==0) && (cap.channels == 3)){
/* Selecciona el canal composite-video (=1) como canal activo. */
 chan.channel=1;
 if (ioctl(fd, VIDIOCSCHAN, &chan) == -1){
  perror("VIDIOCSCHAN");
  close(fd);
  exit(1);
 }
}
```

Para seleccionar el modo de funcionamiento disponemos de nuevos parámetros para la llamada `ioctl`. Con `ioctl(fd, VIDIOCSPICT, &win)` e `ioctl(fd, VIDIOCGPICT, &win)` se nos informa, y se nos permiten ajustar el brillo, contraste, el matiz, la saturación, la profuncidad de la imagen, y el tipo del mapa de colores. Para decir sobre que región vamos a trabajar, si utilizaremos todos los frames, o sólo los pares o impares, utilizaremos `ioctl(fd, VIDIOCGPICT, &vpic)` e `ioctl(fd, VIDIOCSPICT, &vpic)`.

```
if ((strcmp(cap.name,"BT878(Hauppauge new)")==0)||
    (strcmp(cap.name,"BT848A(Leadtek WinView 601)")==0)){
 if (ioctl(fd, VIDIOCSWIN, &win) < 0){
  perror("VIDIOCSWIN");
 }
 if (ioctl(fd, VIDIOCGWIN, &win) < 0){
  perror("VIDIOCGWIN");
  close(fd);
  exit(1);
 }
}
```

```
if ((win.width==mi_ancho)&&(win.height==mi_alto))
/* Caracteristicas de la imagen, incluyendo formato de pixels!!. Se
solicitan al driver y este nos dice que si puede darnoslo o no */
if (ioctl(fd, VIDIOCGPICT, &vpic) < 0){
 perror("VIDIOCGPICT");
 close(fd);
 exit(1);
}
if (win_attributes.depth==24){
 vpic.depth=24;
 vpic.palette=VIDEO_PALETTE_RGB24;
 if(ioctl(fd, VIDIOCSPICT, &vpic)==-1){
  perror("v4l: unable to find a supported capture format");
 }
}
```

Una vez hecho esto, realizaremos la lectura de las imágenes. Ésta se puede hacer empleando una simple llamada al sistema `read()`. Sin embargo este modo de leer imágenes es bastante lento. Cuando se desea capturar a un ritmo mayor es conveniente preparar una zona de la memoria para que el driver deje ahí las imágenes que va capturando. Normalmente estas transferencias se hacen por *DMA* desde la tarjeta a la memoria que le indiquemos. Para que nuestra aplicación pueda leer de la zona de memoria (del kernel) donde el driver recoge las imágenes es necesario solicitarlo al kernel con la llamada `mmap()`. Los parámetros de `ioctl` `VIDIOCSBUF` y `VIDIOCGBUF` inicializan el buffer de captura, y `VIDEOGCCAPTURE` habilita al driver para capturar o lo detiene.

```
gb_buffers.frames=2;
if (cap.type & VID_TYPE_CAPTURE){
/* map grab buffer */
 if (-1 == ioctl(fd,VIDIOCGMBUF,&gb_buffers)){
  perror("ioctl VIDIOCGMBUF");
 }
 map = mmap(0,gb_buffers.size,PROT_READ|PROT_WRITE,MAP_SHARED,fd,0);
 if ((char*)-1 != map){
  printf("Memory mapped: %d Bytes for %d frames\n",gb_buffers.size,
          gb_buffers.frames);
  printf("  BASE=%p\n  OFFSETS: ",map);
  for(i=0;i<gb_buffers.frames;i++) printf("%d ",gb_buffers.offsets[i]);
  printf("\n");
```

```
 }else{
  perror("mmap");
  close(fd);
  exit(-1);
 }
}
/*Lectura de la imagen en la zona de memoria seleccionada para el mapeo*/
for (;;){
 src=map+gb.buffers.offsets[gb.frame];
 ..........
 munmap(map,gb_buffers.size);
}
```

Para finalizar, liberaremos los recursos y cerraremos los dispositivos con la llamada al sistema `close()`.

```
close(fd);
```

3.2. EyeBot

El robot que vamos a utilizar para nuestro comportamiento es el EyeBot. Éste es un robot móvil con el que cuenta el grupo de robótica de la URJC para la creación de un equipo de fútbol que participe en la Robocup.

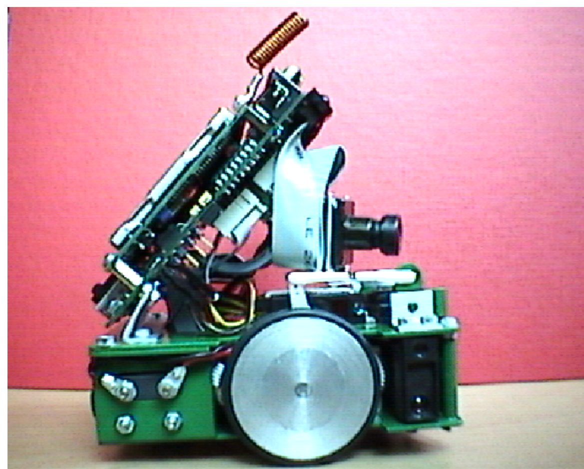

Figura 3.2: Robot Eyebot

El EyeBot dispone de un microprocesador Motorola 68332 a 35 MHz con una memoria Flash-ROM de 512 KB que son para el sistema operativo. Para los programas de usuario tiene una memoria RAM de 1MB que permite ejecutar los programas alma-

cenados en la ROM. Dispone de 2 puertos serie, 1 paralelo, además tiene 8 entradas digitales, 8 salidas digitales y 8 entadas analógicas.

Tiene una pantalla (LCD, Liquid Cristal Display), mediante la cuál es posible visualizar lo que está ocurriendo en el robot. En su parte inferior hay cuatro botones que permiten la interacción del usuario con el programa. Además lleva incluido un micrófono que permite capturar sonidos, y tiene la capacidad de reproducirlos gracias a un altavoz interno

Figura 3.3: Display y botonera

Los elementos de los que dispone para recoger información del entorno son los sensores, infrarrojos, encoders y la cámara. Los infrarrojos miden la distancia a un obstáculo cercano y el EyeBot tiene 3, dos situados a cada lado del robot y otro situado en el frontal. También tiene dos encoders (uno para cada motor), que devolverán un número de pulsos que son indicativos del desplazamiento que cada rueda ha realizado. Y una cámara digital situada en la parte frontal, que captura imágenes del entorno en el que se desenvuelve el robot. Cabe señalar que para nuestro comportamiento no haremos uso de ninguno de estos sensores, ya que nos apoyaremos en la cámara cenital como sensor principal para recoger la información.

El EyeBot, en cuanto a actuadores, cuenta con dos motores para mover las ruedas y dos servos (uno para la cámara y otro para el pateador). Nosotros sólo usaremos los

motores para que el robot se mueva persiguiendo a la pelota.

Además, a través de un puerto serie se pueden descargar los distintos programas al EyeBot. Este puerto también puede ser utilizado para comunicar el robot con un ordenador. Esta comunicación también es posible utilizando el módulo de radio, el cuál permite la comunicación entre distintos robots o con el PC. Este enlace por radio será el que utilicemos para comunicar el EyeBot con el PC, ya que al no utilizar el cable da mayor libertad de movimiento, lo cuál es importante para que el robot pueda seguir a la pelota.

Todo esto que hemos descrito es lo referente a la parte hardware del EyeBot. En cuanto a la parte software, todo EyeBot tiene su propio Sistema Operativo, llamado RoBIOS (Robot Basic I/O System). Éste consta de tres elementos. El primero de ellos es la consola: el sistema operativo gestiona a través de ella los diferentes recursos del sistema y permite al usuario interactuar con el robot. Por ejemplo, permite realizar tests internos a los dispositivos hardware del EyeBot, cargar los programas del usuario, y ejecutarlos. Además es posible ejecutar distintos programas de demostración del funcionamiento del sistema. La operación de la consola se gobierna mediante menús en los que el usuario va eligiendo opciones en pantalla pulsando botones.

Otra parte del sistema operativo es la tabla HDT (Hardware Description Table). En ella se configura el soporte para el diferente hardware conectado. Desde ella se da el nombre a todos los recursos del sistema, y permite a los controladores hardware detectar y usar el hardware conectado al EyeBot. Cabe señalar que a un mismo microprocesador se le pueden conectar diferentes elementos físicos, por ello se puede decir que el EyeBot es un sistema abierto, ya que permite una total configuración de la arquitectura del mismo.

El último elemento del que consta el sistema operativo es una intefaz de programación (API), la cuál facilita a los programas de usuario el acceso a los diferentes recursos. Se compone de un conjunto de funciones para acceder y manipular los distintos sensores, actuadores y el resto de los dispositivos del robot. Estas funciones están escritas en C y se enlazan con el código del programa usuario, de manera que resulta sencillo acceder a los distintos elementos. Por encima de las funciones del sistema operativo, el fabricante proporciona un conjunto de librerías útiles que enriquecen la interfaz para el usuario programador del EyeBot. Por ejemplo la librería VW para el movimiento de los motores, librerías de procesamiento de imágenes, etc. .

Para editar un programa para el EyeBot, lo hacemos en el PC utilizando el lenguaje

ansi-C, por su versatilidad y porque disponemos de un compilador cruzado para la plataforma motorola-68k del eyebot. Una vez escrito el código fuente de nuestro programa lo compilaremos en el PC con el comando `gcc68`. Automáticamente se enlaza con las librerías que ofrece el sistema operativo del EyeBot y se genera el ejecutable .hex. Una vez obtenido el ejecutable se descargará en el EyeBot a través del puerto serie, con el comando `dl`.

Una vez descrita la plataforma de modo genérico, profundizaremos ahora en los detalles de los recursos necesarios para este proyecto, como son los motores y las comunicaciones principalmente.

3.2.1. Actuadores

Los actuadores de los que dispone el robot EyeBot son dos, los motores de continua, los cuáles posibilitan la movilidad del robot. Y los servos, de los que el EyeBot solo tiene dos, uno para mover la cámara y otro para el pateador. A diferencia de los motores de continua, estos pueden ser posicionados y enclavados en un ángulo determinado. Nosotros sólo usaremos los motores de continua.

Hay dos maneras distintas para acceder a los motores, el acceso directo a los mismos (control en lazo abierto) o mediante la librería de control VW (control en lazo cerrado con realimentación). Usaremos el control VW, ya que al tratarse de un sistema realimentado que utiliza los motores junto con los encoders, se obtiene un control más preciso de los movimientos. Las funciones que ofrece el API para la librería de control VW están expuestas en las tablas 3.1 y 3.2.

Con la librería VW, los movimientos del EyeBot pueden controlarse en velocidad o en posición. Nosotros emplearemos el movimiento con control en velocidad, ya que el de posición no nos interesa porque al irse moviendo la pelota, la posición a la que debe ir el EyeBot no es fija. Al utilizar un control en velocidad, éste podrá ser realimentado continuamente según los datos obtenidos de la cámara. Para realizar el control en velocidad disponemos de una serie de funciones, principalmente `VWSetSpeed`, que fija una velocidad objetivo determinada para el EyeBot, tanto lineal como angular. Y si queremos obtener la velocidad actual a la que se está desplazando el EyeBot, usaremos la llamada a `VWGetSpeed`.

Para el uso de este tipo de control se requiere la inicialización del mismo, cosa que se consigue con la llamada a la función `VWInit`, ésta inicializa un manejador para poder utilizar el resto de las funciones de la interfaz. Luego, llamaremos a la función

`VWStartControl` que pone en funcionamiento un controlador que regula la energía suministrada a los motores del EyeBot para controlar su velocidad. Para detener el funcionamiento del controlador, usaríamos `VWStopControl`. Una vez finalizada la utilización de las funciones de la librería VW, es necesario la liberación de los recursos mediante la función `VWRelease`.

3.2.2. Sistema de Comunicaciones

El robot EyeBot dispone de tres dispositivos para comunicarse. Tiene un puerto serie, mediante el cuál podemos conectarlo con el PC para que de ese modo se posibilite la descarga de programas o el envío de comandos, datos y medidas entre distintas plataformas. Estas transmisiones se pueden realizar a diferentes velocidades, 9600, 19200, 38400, 57600, 115200 baudios. También dispone de un puerto paralelo que se encuentra al lado del puerto serie. Se utiliza para conectar el EyeBot con el ordenador y poder realizar una depuración del programa con las herramientas que ofrece Motorola para ello. Por último contiene un módulo de radio mediante el cuál dos o más robots se pueden comunicar entre sí, al igual que un PC y un EyeBot. Esta red trabaja con Token Ring (paso de testigo). Sus características técnicas son: frecuencia de trabajo a 433 MHz, velocidad máxima de transmisión de 9600 baudios, se incluye un protocolo de tolerancia a fallos, transmisión de 8 bits. En nuestro caso, para comunicar el EyeBot con el PC, disponemos de un puente de radio conectado al puerto serie del PC (Figura 3.4).

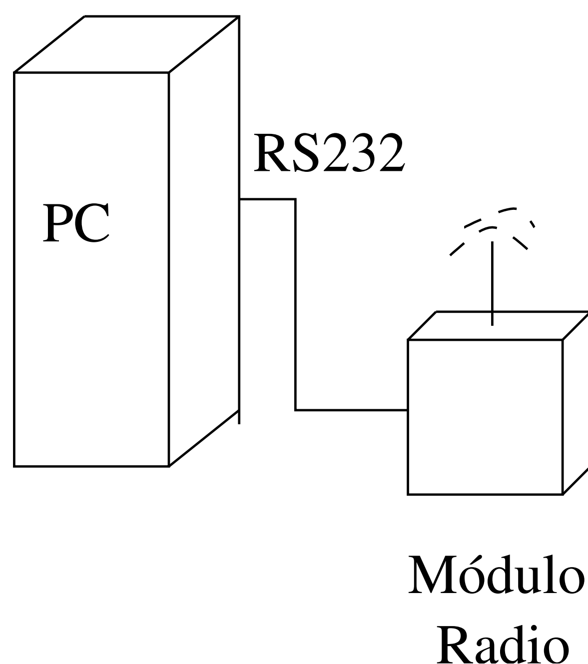

Figura 3.4: Comunicación entre el PC y el EyeBot.

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

Cuadro 3.1: Funciones del interfaz VW

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

Cuadro 3.2: Funciones del interfaz VW (II)

Para la realización de este proyecto solo vamos a utilizar el puerto serie, para la descarga de programas en el EyeBot y el módulo de radio, para la comunicación entre PC y EyeBot en tiempo de ejecución. Para poder comunicar los robots entre ellos, o con el PC, la interfaz de RoBIOS ofrece una serie de funciones. (Ver tabla 3.3).

El uso de las radiocomunicaciones requiere una inicialización previa a través de `RadioInit`. Esta función inicializa y pone en marcha la radio comunicación. Después, es posible tanto un envío como una recepción de mensajes. Para el envío de mensajes, se utiliza la función `RADIOSend`. La longitud máxima del mensaje que se va a enviar no puede superar los 35 caracteres. Para la recepción de mensajes, emplearemos la función `RADIORecv`. Si no hubiera ningún mensaje pendiente, se bloquea el proceso que la ha invocado hasta que llegue el siguiente mensaje. Es pues, una función bloqueante.

Para saber si hay algún mensaje a la espera de ser recibido, usaremos la función `RADIOCheck`. Devuelve el número de mensajes almacenados en el buffer. Se puede utilizar esta función para que el `RADIORecv` no sea bloqueante. Con la función `RADIOGetStatus` podremos averiguar el estado actual de la comunicación, esto es el identificador único del EyeBot y la lista de EyeBots activos en ese momento en la red. Para terminar la comunicación por radio, utilizaremos la función `RADIOTerm`.

Las funciones anteriores, además de ser empleadas en el EyeBot, también se pueden utilizar en el PC. Ya que la librería de radio ha sido también compilada para el PC (`libradio.a`) utiliza las mismas cabeceras que las funciones empleadas en el EyeBot para las comunicaciones. Además, el PC dispondrá de dos funciones más. Una de ellas es `RADIOGetIoctl`, que se encargará de leer la configuración de los parámetros de radio (interface, velocidad, id, remoteOn, imageTransfer y debug). Estos parámetros forman parte de un registro llamdo `RadioIOParameters`. Este resgistro será el parámetro que devuelva la función.

La otra función que emplea el PC es `RADIOSetIoctl`. Ésta cambia la configuración de los parámetros de radio. Deberá ser llamada antes de la llamada a `RADIOInit`. Utiliza el mismo parámetro que la función anterior, pero ahora es de entrada.

3.2.3. Recursos de Multiprogramación

En el EyeBot es posible programar con varias hebras simultánemente, lo cuál es conocido como multiprogramación. Ésta se puede realizar de dos maneras diferentes, con desalojo, en el cuál el control de flujo es independiente de las propias tareas. Periódicamente el sistema operativo desaloja a la tarea actual y le asigna el uso del

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

Cuadro 3.3: Funciones para la radiocomunicación

procesador a otra, de manera que si una tarea se bloquea las restantes podrán seguir ejecutándose. La segunda manera es sin desalojo, también llamado método cooperativo. En este modo sólo se ejecuta aquella tarea que tiene el testigo, de manera que si se bloquea ninguna otra tarea podrá ejecutarse, porque el control de flujo está retenido por la tarea bloqueada, que al bloquearse no puede soltar el testigo.

El método que más se ajusta a nuestras necesidades es el método con desalojo, para que los motores no se detengan mientras la hebra de comunicaciones está bloqueada pendiente de un nuevo mensaje, o viceversa, que no se pueda recibir un mensaje hasta que se acabe cierto movimiento que tiene bloqueada a la hebra de los motores.

El funcionamiento de la multiprogramación en el EyeBot ha de comenzar por una inicialización del entorno de la misma indicando el método deseado, la función `OSMTInit` realiza esta acción. Tras ello se deberán de inicializar cada tarea con una llamada a `OSSPawn`. Esta función devuelve la tarea inicializada e insertada en el planificador, pero no puesta a punto. Para ello se utiliza la función `OSReady`. Una vez que se tengan todas las tareas inicializadas y puestas a punto hay que inicializar el planificador. `OSPermit` realiza esta acción para una multiprogramación con desalojo. Hecho esto, el flujo del programa se distribuye entre las distintas tareas regresando a la función que lanzó el planificador sólo después de que se hayan terminado todas las tareas. Un resumen de las funciones de las que consta la multiprogramación se encuentra en la tabla 3.4.

3.2.4. Interacción del usuario con el EyeBot

Para esta interacción, el EyeBot dispone de tres elementos, la pantalla, la botonera y audio. En primer lugar, dispone de una Pantalla de Cristal Líquido (LCD, Liquid Cristal Display) en la que es posible visualizar lo que está ocurriendo en el robot. Consta de una matriz de 128 x 64 píxeles, en el que se pueden visualizar hasta 17 caracteres ASCII por línea, con un total de 8 líneas en total (la línea inferior está reservada para las etiquetas del menú). El programa que corre en el EyeBot puede acceder a la pantalla para realizar la escritura de cadenas de texto (LCDPrintf (cadena)), de números enteros (`LCDPutInt (entero)`), números reales (`LCDPutFloat(real)`), o el valor hexadecimal de un entero (`LCDPutHex(numero)`). Además será posible representar también las imágenes que se obtiene por la cámara (`LCDPutGraphic(imagen)`), y dibujar líneas (`LCDLine(x1,y1,x2,y2,color)`) y rectángulos (`LCDArea(x1,y1,x2,y2,color)`). Para el borrado total de la pantalla utilizaremos `LCDClear` y para la visualización de 4 cadenas de 4 caracteres cada una, a modo de etiquetas de los botones situados debajo con `LCDMenu(cadena1,cadena2,cadena3,cadena4)`. Las funciones del LCD se describen en las tablas 3.5 y 3.6.

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

Cuadro 3.4: Funciones para las tareas y procesos

En la parte inmeditamente inferior a la LCD hay una hilera de 4 botones cuya funcionalidad dependerá del programa o actividad que se esté ejecutando en cada momento. Hay funciones que permiten la interacción del usuario con el robot a través del teclado, gracias a ellas es posible reconocer la tecla que se ha pulsado (`KEYGet()`). También la función `KEYRead()` que lee la tecla para ver si se pulsó, esta función la usaremos en nuestro proyecto, porque en el programa del EyeBot estaremos a la espera de ver si se pulsa una tecla para la finalización del programa. También podrá detener el flujo de ejecución hasta que se pulse una tecla especificada (`KEYWait(tecla)`). En la figura 3.7 aparecen las funciones del teclado.

El EyeBot también dispone de un micrófono y un altavoz. Éstos posibilitan tanto la grabación como la reproducción de sonidos.

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

Cuadro 3.5: Funciones para la pantalla

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

Cuadro 3.6: Funciones para la pantalla (II)

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

Cuadro 3.7: Funciones para el teclado

Capítulo 4

Descripción Informática

Tras haber descrito los objetivos del proyecto y algunas de las herramientas con las que contamos para conseguirlos en los capítulos anteriores, en éste explicaremos como hemos desarrollado el comportamiento sigue pelota con visión cenital. Empezaremos explicando el diseño conceptual para, a continuación, pasar a contar la implementación del programa y los detalles de los algoritmos de percepción y control empleados.

4.1. Diseño del Comportamiento

Como ya comentamos anteriormente, un requisito para realizar el comportamiento sigue pelota con visión cenital es que ha de seguir la estructura de la arquitectura JDE [Cañas02]. En esa línea se diseñó el comportamiento sigue pelota como la ejecución simultánea de dos esquemas, uno perceptivo y otro motriz. El perceptivo es el encargado de analizar las imágenes obtenidas por la cámara cenital, con la finalidad de distinguir en ellas al EyeBot y a la pelota. Recibe como entrada la imagen capturada por la cámara cenital y da como resultado el ángulo y la distancia relativos de la pelota respecto del EyeBot. El esquema motriz recibe como entrada los datos de salida del esquema perceptivo, es decir, el ángulo y la distancia que forman robot y pelota. A partir de estos datos, mueve los motores con el fin de que el robot siga a la pelota. Este diseño conceptual, lo podemos ver en la figura 4.1.

Para implementar estos esquemas hacemos uso de dos programas escritos en C, `ebase.c` y `emovil.c`. El `ebase.c` se ejecuta en el PC y el `emovil.c` en el EyeBot. Como podemos ver, este comportamiento no se ha implementado en su totalidad en el robot, como sí ha ocurrido con otros proyectos tales como el del sigue pelota con vision local [SanMartín02] y el sigue pared [Gomez02]. Este proyecto está más orientado al planteamiento clásico de la liga pequeña de Robocup, en el que el PC es el encargado de hacer todos los procesamientos. En el `ebase.c` se capturan las imágenes y se analizan

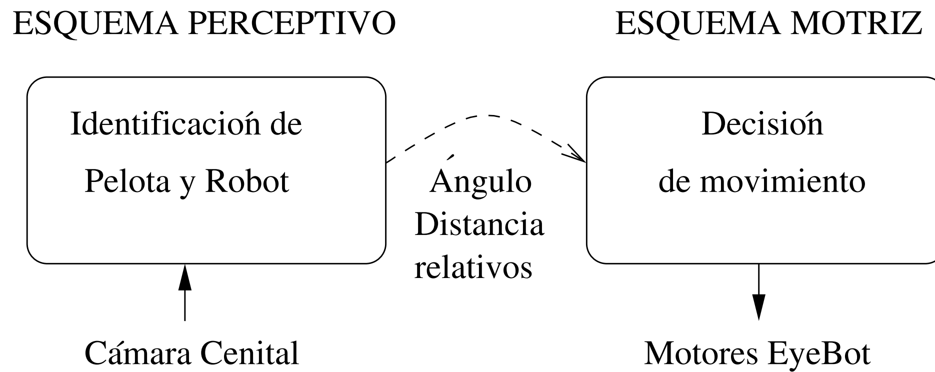

Figura 4.1: Comportamiento con dos Esquemas.

para identificar las posiciones del EyeBot y de la pelota. Con estas posiciones calcula el ángulo y la distancia relativas que forman entre ellos. Con estos datos se realiza una toma de decisiones de movimiento adecuado, las cuáles se materializan en un mensaje que será enviado por medio de la comunicación por radio al EyeBot. De este modo, en el `ebase.c` se han implementado dos esquemas, el perceptivo, y parte del esquema motriz.

El programa `emovil.c` que se ejecuta en el robot, recibe este mensaje y trás analizarlo mueve los motores de acuerdo a los datos recibidos para que se cumpla el comportamiento.

Para separar las dos partes del `ebase.c` (análisis de la imagen y decisión de control) podríamos haber usado dos hebras, pero debido al hecho de que las funciones `sleep`, `usleep` y `nanosleep` dejaban de funcionar correctamente por la captura de la señal `SIGALARM` (como veremos en la sección referente a la comunicación por radio), era imposible que esas hebras ejecutaran iteraciones periódicas, ya que este periodo se controla mediante el empleo de estas funciones.

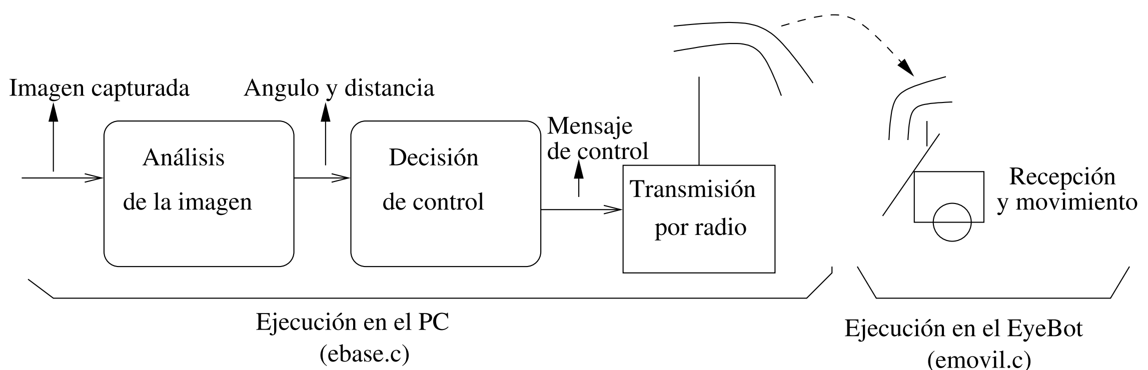

Figura 4.2: Estructura del comportamiento sigue pelota con visión cenital.

4.2. Esquema Perceptivo

Como ya hemos explicado, el esquema perceptivo está implementado en el programa `ebase.c` que se ejecuta en el PC. Es el encargado de analizar las imágenes capturadas por la cámara cenital con la finalidad de identificar la posición del EyeBot y de la pelota en el terreno de juego.

Para analizar la escena hemos utilizado cuatro etapas. Una primera fase, puramente sensorial, que consiste en la captura, o adquisición de las imágenes digitales por medio de la cámara cenital, para ello, utilizamos la interfaz video4linux, tal y como describimos en el capítulo 3. En la segunda etapa pasamos un filtro de color a la imagen, en el espacio HSI, para realzar las zonas en donde se encuentran la pelota y el EyeBot. Para poder utilizar este filtro hemos utilizado unos colores llamativos: la pelota es de color naranja y el EyeBot lleva dos pegatinas de distintos colores, una azul y otra verde. Usamos dos pegatinas para que, además de saber la posición del robot en el campo de juego, sepamos también su orientación, es decir, hacia donde mira. Esta orientación nos la da la pegatina azul. En la tercera fase, realizamos una segmentación, basada en histogramas, que aisla los elementos que nos interesan de la imagen agrupando los píxeles de un determinado color. Y en la última etapa distinguimos, a partir de los objetos segmentados, en qué coordenadas está situada la pelota y el EyeBot.

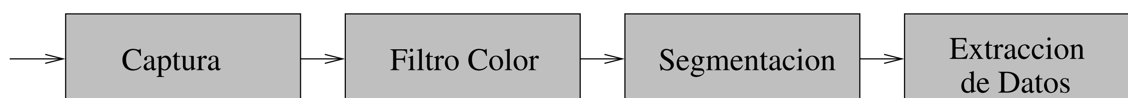

Figura 4.3: Fases para el análisis de la imagen

Con las posiciones obtenidas hallamos la distancia y el ángulo formado entre la pelota y el robot. Las cuáles son utilizadas en el esquema motriz como información básica para poder saber cuánto debe girar y a qué velocidad debe avanzar el EyeBot.

4.2.1. Captura

Las imágenes digitales son "señales" discretas, que suelen tener origen, normalmente, en una "señal" continua. En el proceso de obtención de imágenes digitales con cámaras analógicas se distinguen dos etapas. La primera, conocida como *captura*, utiliza un dispositivo óptico, con el que obtiene información relativa a una escena. En nuestro caso, el dispositivo será la cámara cenital. En la segunda etapa, que se conoce como *digitalización*, se transforma esta señal continua de vídeo, en la imagen digital, que es una señal discreta.

Para la captura de imágenes hemos utilizado una cámara analógica `CCD de SUN` y una tarjeta digitalizadora `BT878`. Como mencionamos en el capítulo 3, esas imágenes se ponen accesibles a nuestro programa a través de la interfaz video4linux. Cabe señalar que el ritmo de captura (sin analizar la imagen) que obtenemos es aproximadamente a 10 imágenes por segundo. En la figura 4.4 podemos ver un ejemplo de una imagen capturada por la cámara, en la que se ve el terreno de juego, y sobre él está la pelota y el EyeBot.

Figura 4.4: Imagen capturada por la cámara cenital

4.2.2. Filtrado

Como hemos comentado, en esta etapa lo que pretendemos es realzar las zonas en donde se encuentran la pelota y el EyeBot. Para realizar esto, hemos empleado un filtro de color. Realizamos un filtro para realzar las partes en la imagen que posteriormente vamos a analizar (ver figura 4.5).

Inicialmente, cuando capturamos la imagen, ésta se encuentra en un formato RGB, ya que de esa manera hemos configurado video4linux en el `ebase.c`. En éste, cada píxel ocupa tres bytes correspondientes a la terna RGB: un byte es para la componente roja (Red), otro para la verde (Green) y un tercero para la azul (Blue). En las primeras implementaciones, puesto que la imagen estaba en este formato, decidimos aplicar el filtro de color en este espacio (RGB). Tras varias pruebas, nos dimos cuenta de que este filtro no era nada robusto frente a cambios en la luminosidad, ya que los resultados variaban dependiendo de que utilizáramos la luz proveniente de las ventanas o bien que empleáramos la luz artificial. Por ello, decidimos realizar un filtro en el espacio HSI, y pudimos comprobar, tras varias pruebas, que es mucho más robusto que el anterior. Ahora bien, como la imagen se encuentra en el formato RGB, tenemos que convertirla al formato HSI. En este último, cada píxel tiene 3 componentes HSI, Matiz

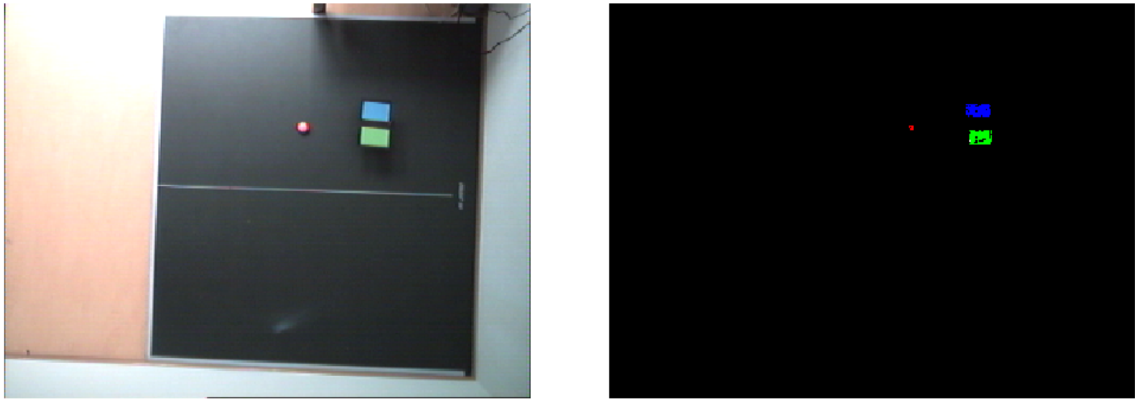

Figura 4.5: Imagen obtenida por la cámara cenital (Izquierda) e imagen filtrada (Derecha).

color	Hmax	Hmin	Smax	Smin
Pelota Naranja	0.2	0.4	0.2	0.4
Etiqueta Verde	2.0	2.4	0.1	0.22
Etiqueta Azul	2.6	3.15	0.22	0.5

Cuadro 4.1: Rango de valores HS para la pelota y el EyeBot

(H), Saturación (S) e Intensidad (I). Para obtener el formato HSI, lo que hacemos es aplicar las siguientes fórmulas a cada píxel:

$$H = \arccos\left(\frac{\frac{(R-G)+(R-B)}{2}}{\sqrt{(R-G)^2+(R-G)(G-B)}}\right)$$

$$S = 1 - \left(\frac{3}{(R+G+B)}\right)[min(R,G,B)]$$

$$I = \frac{1}{3}(R+G+B)$$

en donde R es la componente roja, G la verde y B la azul.

Dicho esto, el filtrado consiste en comprobar píxel a píxel que las componentes correspondientes al matiz y a la saturación están comprendidas en un cierto rango de valores. La componente de la intensidad no la comparamos porque no nos interesa, ya que los cambios de luminosidad se reflejan principalmente en la componente I y nuestro filtro pretende ser robusto frente a estos cambios. Por ello debemos ignorar esta componente. El rango de valores para el Matiz y la Saturación de la pelota y del EyeBot, está representado en la tabla 4.1. Estos valores los hemos ajustado experimentalmente para los colores buscados en el escenario del laboratorio.

Una vez realizado el filtro a la imagen, para ayudar a la depuración del programa, se muestra una interfaz gráfica compuesta por dos ventanas, en una se puede ver la imagen capturada, y en la otra la imagen filtrada y segmentada (Figura 4.5). En la imagen

filtrada, los píxeles que están dentro del rango que consideramos naranja, los hemos representado con el color RGB #FF0000, los que estaban dentro del rango del verde, tienen el color RGB #00FF00, y por último los comprendidos en el rango del azul, se representan por el color RGB #0000FF. El resto de píxeles que no entran en ninguno de estos 3 rangos, tienen el valor negro RGB #000000. Cabe señalar que dependiendo de la configuración de las XWindows, el formato de visualización puede variar, ya que la profundidad de la imagen cambia. Nuestro programa soporta la visualización de estas imágenes en servidores X con 16 bits de profunidad y con servidores de 32 bpp. Cuando profundidad es de 16 bits, tenemos dos bytes por cada píxel en el que las componentes RGB se distribuyen según muestra la figura 4.6. La componente roja corresponde a los 5 bits más significativos del segundo byte, la componente verde corresponde a los 3 bits más significativos del primer byte más los tres bits menos significativos del segundo byte. Y la componente azul consta de los cinco bits menos significativos del primer byte.

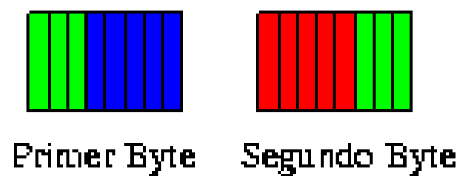

Figura 4.6: Formato del color para una configuración de 16 bits

En cambio, para profundidad 32 bits, cada píxel tendrá 4 bytes, el primer byte corresponde a la componente azul, el segundo a la verde, el tercero a la roja, y el cuarto no contiene valor significativo alguno, según muestra la figura 4.7.

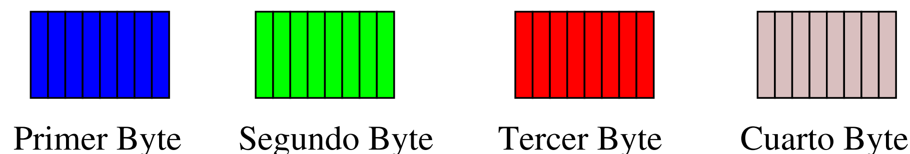

Figura 4.7: Implementación del color para una configuración de 32 bits

4.2.3. Segmentación

La segmentación es un proceso que consiste en dividir una imagen digital en regiones homogéneas con respecto a una o más características, en nuestro caso el color, con el fin de facilitar su posterior análisis y reconocimiento. La técnica que emplearemos para segmentar, es una basada en histogramas. Un histograma es un diagrama de barras que contiene frecuencias relativas a alguna característica. En nuestro caso por

cada objeto que queramos segmentar, tenemos dos histogramas, uno para las columnas (histograma X) y otro para las filas (histograma Y). En el histograma X, tenemos representado en el eje de abcisas, cada columna. Y en el eje de ordenadas, la frecuencia con la que aparece un píxel de un determinado color (la pelota tendrá el color rojo, la pegatina verde, el color verde y la pegatina azul, el color azul) en una columna (Figura 4.8). El histograma Y es homólogo, con la salvedad que en el eje de abcisas estará representadas las filas en vez de las columnas (Figura 4.8). Debemos calcular ambos histogramas para cada color que estamos buscando. El motivo por el que hallamos estos histogramas, es para enventanar las zonas con alta densidad de píxeles relevantes, para agrupar las zonas que son naranja, verde y azul.

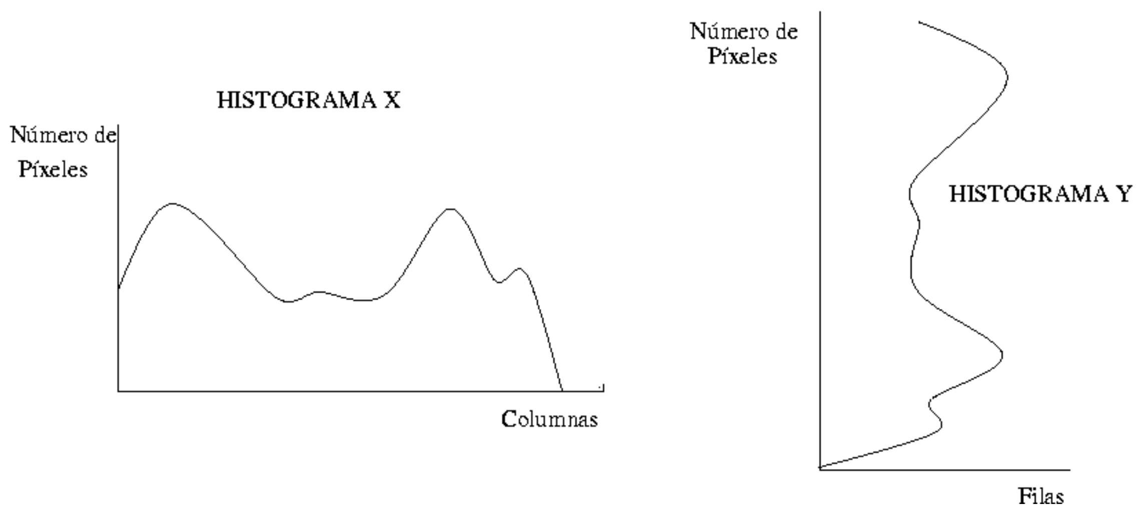

Figura 4.8: A la izquierda el histograma en el eje X y a la derecha el histograma en el eje Y.

Una posible técnica de segmentación desde el histograma consiste en utilizar un umbral simple, pero descartamos esta idea debido a que utilizar un umbral simple acarrea dos problemas. Por un lado el poner un umbral alto quitaría parte de los objetos, y por el otro, uno bajo cogería demasiado ruido (Figura 4.10). Por ello decidimos aplicar un doble umbral a los histogramas [SanMartín02](Figura 4.9). Cabe señalar que la razón por la que hemos realizado este tratamiento independiente a cada eje, es para que el proceso de segmentación se efectúe de manera más rápida y eficiente.

Para obtener los límites iniciales de cada zona, apuntamos la columna o fila que supera el primer umbral, siendo el fin de la zona el punto en donde se pasa por debajo de ese umbral. Vamos cogiendo las restantes zonas hasta el final del histograma. Esto nos da varias zonas, pero solo nos quedamos con aquellas zonas en las que en algun momento, ademas de pasar el primer umbral, pase también un segundo. En la figura 4.9 podemos ver como 3 zonas pasan el primer umbral, pero solo dos además pasan el

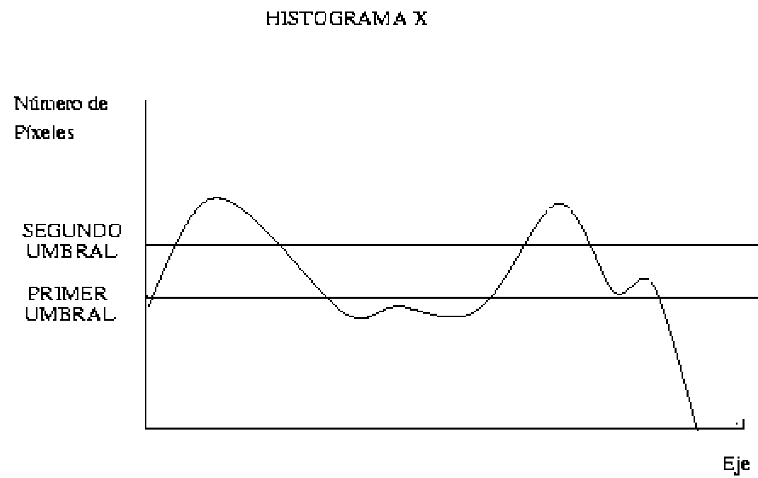

Figura 4.9: Histograma al que se le ha realizado un Doble Umbral.

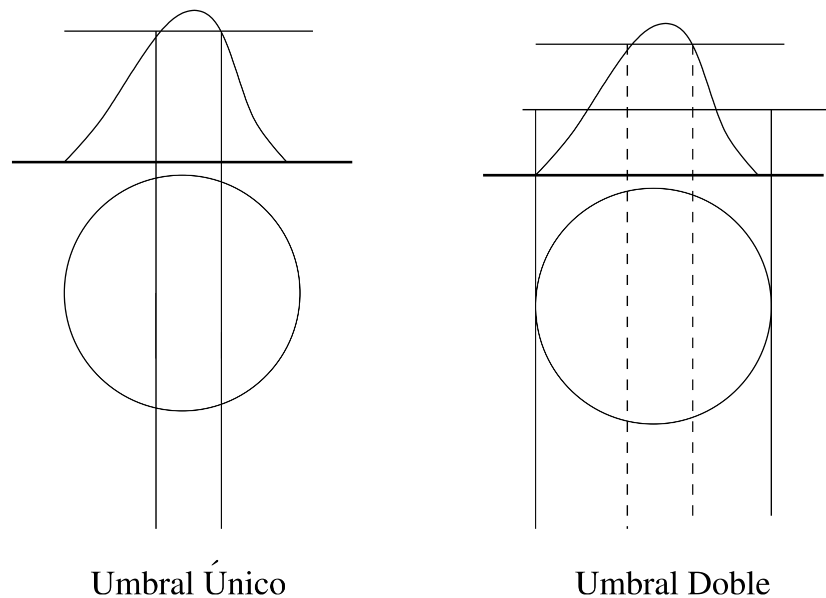

Figura 4.10: Comparativa entre usar un umbral único y uno doble.

segundo, por lo que sólo nos quedamos con esas 2. El segundo umbral asegura que hay un número relevante de píxeles del color que andamos buscando.

Este algoritmo lo pasamos tanto en el histograma X como en el Y, para hallar los puntos que definen cada ventana, tal y como aparece en la figura 4.11. Como podemos ver en esa figura, podemos obtener varias ventanas y alguna de ellas vacías (ventanas fantasmas), para solucionar este problema, nos quedamos con aquella ventana que posea el mayor número de píxeles del color que andamos buscando o al menos un número significativo de píxeles de ese color en su interior. Podemos ver un ejemplo de una imagen resultante tras la segmentación en la figura 4.12, donde se han pintado los contornos de las ventanas, además de los píxeles de color.

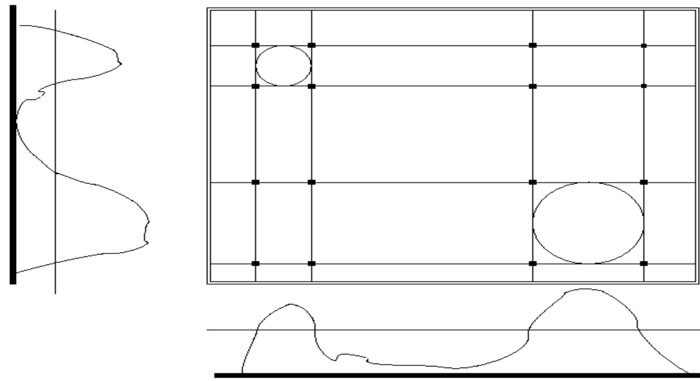

Figura 4.11: Ejemplo del eventanado

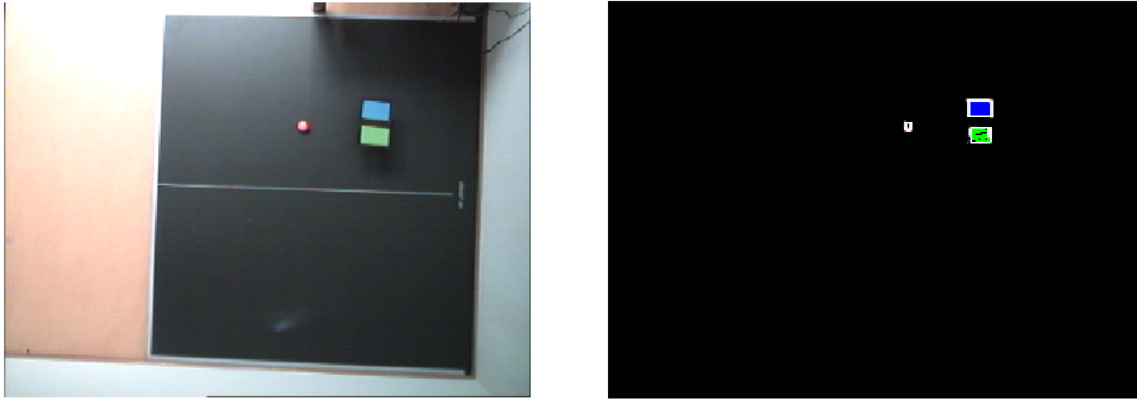

Figura 4.12: Imagen obtenida por la cámara cenital (Izquierda) e imagen segmentada (Derecha).

4.2.4. Identificación de la pelota y del robot

Una vez segmentada la imagen obtendremos, en el caso normal, tres ventanas (figura 4.12). Una que encuadra la pelota, y otras dos referentes al EyeBot, una por cada pegatina. A continuación hallamos el centro de cada ventana. Cada ventana tiene cuatro vértices, $P_1(x_{min}, y_{min})$, $P_2(x_{max}, y_{min})$, $P_3(x_{min}, y_{max})$ y $P_4(x_{max}, y_{max})$. Entonces para hallar la coordenada centro, lo que hacemos es: $P_{centro}(\frac{x_{min}+x_{max}}{2}, \frac{y_{min}+y_{max}}{2})$. Con cada coordenada centro, de cada ventana, obtenemos la posición de la pelota y la de las dos pegatinas. Para hallar la posición del EyeBot, hallaríamos la coordenada del punto medio entre ambas pegatinas. Si la pegatina azul tiene por coordenadas $A(x_A, y_A)$ y la pegatina verde tiene por coordenadas $V(x_V, y_V)$, la posición del EyeBot tal y como hemos situado las pegatinas en el robot es: $E(\frac{x_A+y_V}{2}, \frac{y_A+y_V}{2})$.

Además de obtener las posiciones del EyeBot y de la pelota, nos interesa también hallar el ángulo y la distancia entre ellos, porque estos datos nos servirán, dentro del esquema motriz, para saber hacia donde ha de girar o avanzar el EyeBot, y a qué velocidad hacerlo. Tenemos tres puntos relevantes, la coordenada centro de la pegatina azul (A), la coordenada referente a la posición del EyeBot (E) y la de la pelota

(P). Como un vector está determinado por dos puntos, obtendremos dos vectores. El vector del EyeBot, que está representado por las coordenadas A y E; y el vector que forman la pelota y el EyeBot, determinado por los puntos P y E. Si tenemos $A(x_a, y_a)$, $E(x_e, y_e)$ y $P(x_p, y_p)$, el vector del EyeBot será $\vec{EA}(x_a - x_e, y_a - y_e)$ y el vector EyeBot pelota es $\vec{EP}(x_p - x_e, y_p - y_e)$, tal y como muestra la figuar 4.13.

Para hallar el valor de la distancia entre el EyeBot y la pelota, calculamos el módulo del vector formado por ambos, ya que el módulo de un vector representa la longitud del mismo.

$$distancia = \left|\vec{EP}\right| = \sqrt{x_{EP}^2 + y_{EP}^2}$$

Una vez hallada la distancia, calculamos el ángulo formado entre el robot y la pelota. Para hallarlo, podríamos haber empleado la fórmula del producto escalar, el cuál es el número real que resulta al multiplicar el producto de sus módulos por el coseno del ángulo que forman

$$\vec{EA}\vec{EP} = \left|\vec{EA}\right| \left|\vec{EP}\right| \cos(\widehat{\vec{EA}, \vec{EP}})$$

de donde deducimos que:

$$\widehat{EA, EP} = \arccos\left(\frac{\vec{EA}\vec{EP}}{\left|\vec{EA}\right| \left|\vec{EP}\right|}\right) \qquad \forall \vec{EA}, \vec{EP} \neq \vec{0}$$

Pero esta fórmula nos ofrecía un problema, que el valor devuelto correspondía al valor absoluto del ángulo, por lo que nos resultaba difícil el poder calcular el sentido de giro y necesitábamos realizar un cálculo adicional. Por ello, decidimos tomar otra alternativa.

Para ello, calculamos los ángulos que forman los vectores $\vec{EA}$ y $\vec{EP}$ con el eje de abcisas (figura 4.13). Al ángulo formado por $\vec{EA}$ con el eje de abcisas, lo llamaremos α_1, y al formado por $\vec{EP}$ con este mismo eje, lo llamaremos α_2. Para calcular α_1 y α_2, lo que hacemos es lo siguiente: en el caso de α_1 tomamos como centro de coordenadas al punto E. Calculamos la proyección de A sobre el eje de abcisas A'. Con estos 3 puntos, E, A y A' formamos un triángulo rectángulo, y de ahí, deducimos las siguientes fórmulas:

$$sin(\alpha_1) = \frac{y_A - y_E}{EA}$$

$$cos(\alpha_1) = \frac{x_A - x_E}{EA}$$

$$tag(\alpha_1) = \frac{y_A - y_E}{x_A - x_E}$$

Usamos la fórmula de la tangente, ya que tenemos los valores de x e y, que coinciden con las coordenadas del vector $\vec{AE}(x_{AE}, y_{AE})$. Para hallar α_2 hacemos lo mismo, con la

salvedad de que los tres puntos ahora serían E, P, P'. Hemos utilizado la función `atan2` de la librería `math.h` de C en vez de `atan` (también de la misma librería), porque ubica perfectamente el ángulo sin incertidumbres. Esto es así porque la función `atan2` calcula el valor principal del arcotangente $\frac{y_A - y_E}{x_A - x_E}$, utilizando los signos de ambos argumentos para determinar el cuadrante del valor devuelto. Devuelve el arco tangente de $\frac{y_A - y_E}{x_A - x_E}$ en el rango $[-\pi, +\pi]$ radianes. En cambio, la función `atan`, calcula el valor principal del arco cuya tangente es $x_A - x_E$. Devuelve el arco tangente situado en el rango de $[\frac{-\pi}{2}, \frac{+\pi}{2}]$ radianes.

Una vez calculados sendos ángulos, α_1 y α_2, para calcular el ángulo α que forman ambos, restamos $\alpha_1 - \alpha_2$. Si el ángulo resultante es mayor que 180°, lo que hacemos es restar $\alpha - 360°$, de igual modo, si el ángulo es mayor que -180°, sumamos $360° + \alpha$. El sumar o restar 360° mantiene exactamente igual el ángulo y permite acotar los valores posibles entre ±180°. Si al final, α fuera mayor o igual que 0, significa que la pelota está situada a la izquierda del EyeBot, por el contrario, si fuera menor que cero, estaría situada a la derecha.

Cabe mencionar, que tras el análisis de la imagen, con filtrado, segmentado y cómputo, el ritmo de imágenes por segundo resultante es aproximadamente 7.

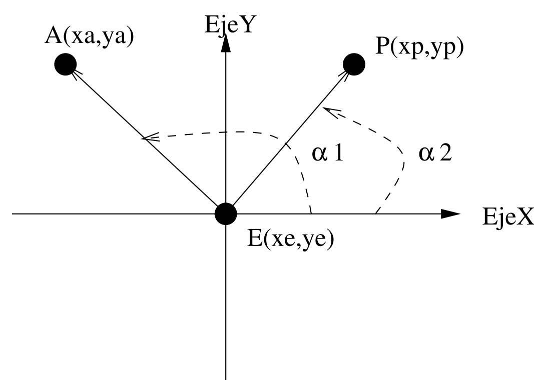

Figura 4.13: Ángulos de dos vectores con el eje de abcisas.

4.3. Esquema Motriz

Este esquema toma como datos de partida, el ángulo y la distancia entre el EyeBot y la pelota. Con estos datos se deberá tomar unas decisiones de control adecuadas para que el EyeBot pueda seguir a la pelota. Para realizar esto, vamos a dividir este esquema en tres partes. Primeramente la parte de control en la que se toma de decisión de a qué velocidad y cuánto debe girar el EyeBot. En segundo lugar, formamos el mensaje en el que indicaremos la anterior información. Y lo enviaríamos

por radio desde el PC al robot. Y, la ultima parte, sería la recepción del mensaje por parte del EyeBot y la materialización de sus órdenes en él.

Para implementar el esquema motriz, hemos utilizado dos programas, el `ebase.c` que corre en el PC, y ejecuta las dos primeras partes, y el `emovil.c` que se ejecuta en el EyeBot, e implementa la tercera parte.

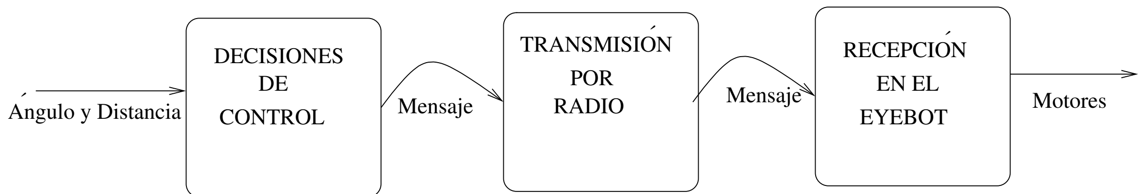

Figura 4.14: Partes del Esquema Motriz

4.3.1. Decisiones de Control

Esta parte consiste en la determinación de los comandos motores adecuados para que el EyeBot pueda seguir a la pelota. Para tomar estas decisiones de movimiento, tenemos que tener en cuenta los datos obtenidos al analizar la imagen, como son el ángulo α y la distancia d entre el EyeBot y la pelota, y decidir en función de ellos el movimiento adecuado según el comportamiento sigue pelota. Este movimiento se materializa en un control en velocidad del robot (v y ω).

Primeramente comprobamos el valor absoluto del ángulo α. Si éste se encuentra entre 0° y 30°, podemos decir que el robot se encuentra bien orientado frente a la pelota, por lo que no le deberíamos indicar velocidad de giro alguna. En esta situación, comprobamos si la distancia está entre los valores 0 y 150 píxeles, si es así, nos encontraríamos frente a la pelota, y le diríamos al robot que se pare. Si la distancia es superior a 50 píxeles, le mandaríamos sólo velocidad de tracción proporcional a la distancia, para que el EyeBot se dirija hacia ella. (Figura 4.15, imagen derecha).

En segundo lugar, si está situado entre 30° y 150°, le indicaríamos una velocidad de giro proporcional al valor del ángulo α ($\omega = 0{,}02\alpha$) y una pequeña velocidad de tracción proporcional a la distancia ($v = (0{,}3/100)d$).

Por último, si el valor absoluto de α es superior a 150°, le daremos una velocidad de giro fija ($\omega = 1{,}0$) y, como antes una velocidad de tracción proporcional a la distancia ($v = (0{,}1/100)d$).

Cuando el valor absoluto del ángulo sea mayor a 30°, si el robot se encontraba parado, le añadiremos un pequeño pico a la velocidad de tracción. Sin este pico, el robot no avanza aunque se le comande cierta velocidad porque no supera el rozamiento estático.

Cabe señalar, que si la pelota no esta situada en el terreno de juego, le mandaríamos una velocidad de tracción y de giro nulas.

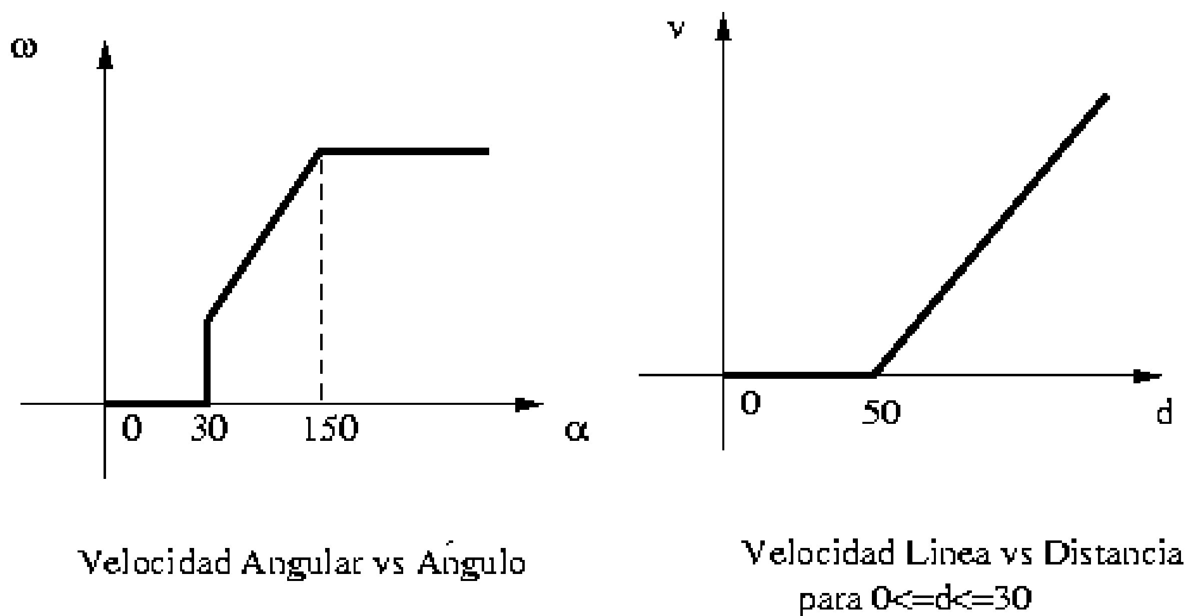

Figura 4.15: Perfiles de control, Velocidad Angular vs Ángulo (Izquierda) y Velocidad Lineal vs Distancia (Derecha)

4.3.2. Transmisión por Radio

Una vez tomadas las decisiones de control, formaremos un mensaje que enviaremos al EyeBot. Este mensaje tiene como cabecera la palabra COMMANDED, y a continuación, separados por blancos, indicaríamos la velocidad de tracción primero y después la de giro. Por ejemplo, para comandar 0.5 m/s y 1.0 rad/s, le enviaríamos:

```
COMMANDED 0.5 1.0
```

Tras la formación del mensaje, debemos enviarlo desde el PC al EyeBot mediante la comunicación por radio. Por ello, hemos tenido que establecer una comunicación fiable entre el PC y el EyeBot, puesto que la librería del fabricante (descrita en el capítulo 3) no ofrece esta fiabilidad. Uno de los problemas importantes que tuvimos fue que al intentar mandar muchos mensajes seguidos al robot (*streaming*), se saturaba el búffer de comunicaciones, y dejaba de envíar mensajes. Intentamos solucionarlo poniendo un

tiempo de espera con la llamada a `usleep()` entre cada mensaje, es decir, intentamos controlar el ritmo emisor para que no se saturara el búffer. Pero la librería libradio.a captura la señal `SIGALARM`, señal en la que se basa la función `usleep`, que por ello dejaba de funcionar correctamente.

Tuvimos que buscar otra manera de solucionarlo, ya que no había manera de poner un tiempo de espera entre cada mensaje. Por ello decidimos crear dos procesos dentro del `ebase`. Estos dos procesos se comunican a través de un pipe. El proceso Padre sirve como temporizador, controla el ritmo al que se emite los mensajes. Para ello escribe en la entrada del pipe un carácter cada 100 milisegundos, para avisar al proceso Hijo que puede envíar el mensaje. Este retardo se justifica porque 10 mensajes/segundo es un número aceptable por el enlace de radio y supone una cantidad de iteraciones de control suficiente para un buen movimiento. El proceso Hijo comprobará, antes de emitir un mensaje, que se cumplan dos condiciones, (1) que se haya formado un mensaje para enviar y que, además, (2) haya algo que leer en la salida del pipe. Con esto, solucionamos la captura de la señal `SIGALARM`, ya que al tener dos procesos, el Padre activa el `usleep` para esperar cien milisegundos, y el Hijo inicializa la radio. Al ser independientes ambos, la librería de radio no afecta al empleo de la llamada al sistema `usleep` en el Padre.

En el otro proceso no arreglábamos la saturación del búffer, ya que si pasaba una décima de segundo y no se analizaba una imagen en ese periodo de tiempo (porque en ese momento la CPU estaba ocupada), se podían acumular caracteres en la salida del pipe. Esto implicaba que si dos imágenes eran analizadas antes de que pasaran 100 milisegundos, al haber varios caracteres en el pipe, se podían enviar dos mensajes sin haber esperado este período de tiempo. Por ello, trás enviar el mensaje por la radio al EyeBot, deberemos leer en cada iteración todos los caracteres disponibles en el pipe para evitar acumulación de caracteres.

Cabe señalar que con esto hemos controlado el flujo tanto si el análisis de las imágenes es muy lento (por lo que iríamos al ritmo del análisis) o, por el contrario, es muy rápido (por lo que vamos al ritmo del reloj). El reloj marca el ritmo máximo de envío.

```
err=pipe(tubo);
if (err == -1) printf ("Pipe error\n");
mychild = fork();
if (mychild!=0)
{
```

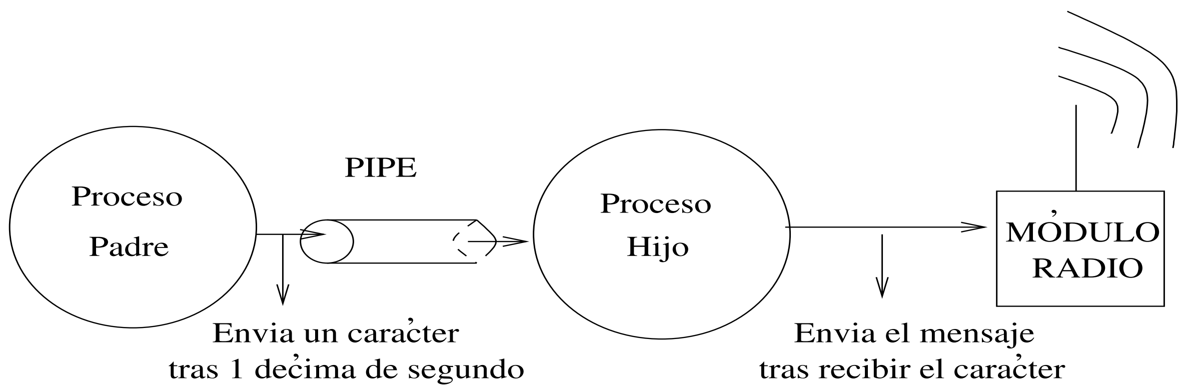

Figura 4.16: Transmisión del mensaje por la radio

```
 signal(SIGTERM, NULL); /* kill interrupt handler */
 signal(SIGINT, NULL); /* control-C interrupt handler */
 close(tubo[0]);
 for(;;)
 {
  usleep(100000);
  err=write(tubo[1],&a,1);
  if (err == -1) printf ("Writing in pipe error\n");
 }
}
else
{
if(fcntl(tubo[0], F_SETFL, O_NONBLOCK) < 0)
/* Lo pone NO BLOQUEANTE  */
{ printf("fcntl FSETFL, O_NONBLOCK");  exit(1);}

 close(tubo[1]);
 RADIOGetIoctl(&radioParams);
 radioParams.speed = SER38400;
 radioParams.interface = SERIAL1;
 RADIOSetIoctl(radioParams);

 err = RADIOInit();
 if (err)
 {
  printf("Initing radio error\n");
 }
 printf ("Radio started\n");
```

```
  for(;;){

  .....................
  err=read(tubo[0],&c,1);
  if (err>0)
  {
   sprintf(output_buffer,"%d %1.1f %1.1f\n",COMMANDED,
   commanded_trans_vel,commanded_rot);
   printf("(%d):%s",n++,output_buffer);
   err = RADIOSend(1, strlen(output_buffer),output_buffer);
   if(err){
    printf("Sending error\n");
    RADIOTerm();
    while (err){
     printf ("inicializando\n");
     err = RADIOInit();
     if (err) printf("error al inicializar\n");
    }
   }
   while (err>0) {err=read(tubo[0],&c,1);}
  }
 }
}
```

4.3.3. Eyebot

El `emovil.c` es el programa receptor que se ejecuta completamente en el EyeBot. Es el que materializa las órdenes de movimiento que decide y envía `ebase.c` al robot. Consta de tres hebras, `‘‘Recibe Puerto’’`, `‘‘Mueve Motores’’` y `‘‘Watchdog’’` (Figura 4.17).

`‘‘Recibe Puerto’’`, es la encargada de esperar a que le llegue un mensaje proveniente del PC. Una vez recibido, lo analizará. Para ello, primeramente comparará la cabecera. Si ésta es igual a `‘‘ADIOS’’`, parará los motores. Por el contrario, si la cabecera es `‘‘COMMANDED’’`, analiza el cuerpo del mensaje para hallar las órdenes de velocidad de tracción y de giro. Estas velocidades son guardadas en dos variables, las cuáles las usa también la segunda hebra, `‘‘Mueve Motores’’`.

```
void recibe_puerto()
```

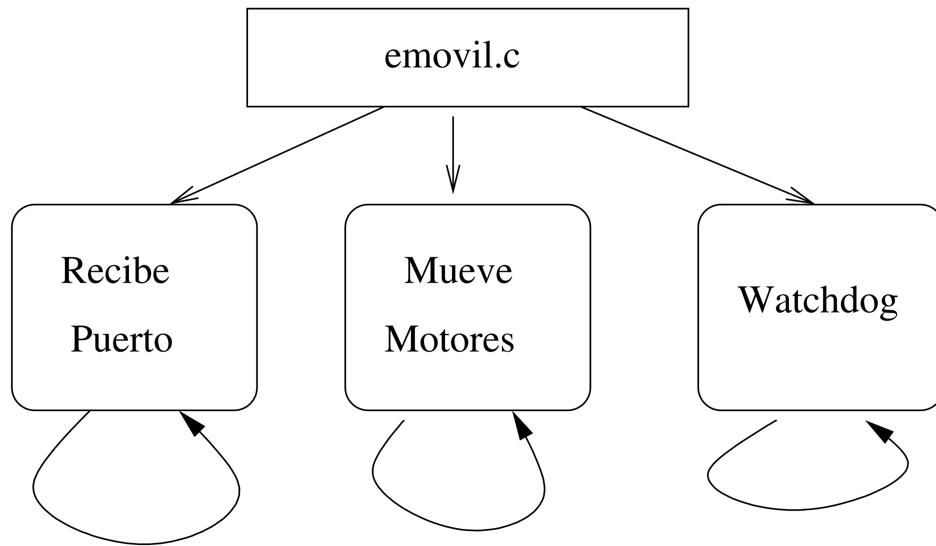

Figura 4.17: Hebras del emovil.c

```
/* communication thread */
{
 BYTE buffer[MESSAGE_SIZE];
 int marca=0;
 int z=0;

 for (;;){
  if (RADIOCheck())
  {
   RADIORecv(&fromId,&len,buffer);
   z=0;
   while (z<len){
    if (buffer[z]=='\n'){
     input_buffer[marca]='\0';
     sscanf(input_buffer,"%d %s",&cabecera, contenido);
     LCDPrintf("(%d)\n",receivedmessages);
     if (cabecera == ADIOS){
      VWSetSpeed(vw,0,0);
      drive=0.; steer=0.;
      LCDPrintf("Bye\n");
     }
     if (cabecera == COMMANDED){
      sscanf(input_buffer,"%d %f %f",&cabecera,&drive,&steer);
      receivedmessages++;
      watchdog_on=0;
```

```
      if (drive > MAX_V) drive=MAX_V;
      else if (drive < -MAX_V) drive=-MAX_V;
      if (steer > MAX_W) steer=MAX_W;
      else if (steer < -MAX_W) steer=-MAX_W;
     }
     marca=0;
     z++;
    }else{
     input_buffer[marca]=buffer[z++];
     marca++;
    }
   }
  }
 }
}
```

``Mueve Motores'' es la hebra responsable del movimiento del robot según la orden que haya recibido del emisor. Dentro de esta hebra, introducimos una espera para ceder más tiempo de CPU a la hebra de ``Recibe Puerto'', ya que en caso contrario, esta hebra bloquea las comunicaciones porque la hebra ```Recibe Puerto'' no dispone de suficiente tiempo de CPU para su ejecución.

```
void mueve_motores()
/*thread to move motors according to received messages */
{
 while(1){
  VWSetSpeed(vw,drive,steer);
  OSSleep(4);
  /* this adhoc delay is required to let the communication thread use
     enough computer time */
 }
}
```

La última hebra es ``watchdog''. Es un perro de guarda que chequea cada cada medio segundo si se han recibido nuevas órdenes de movimiento. Si no se ha recibido ningún nuevo mensaje detiene el movimiento del robot como medida de seguridad. Esta protección es necesaria para evitar choques si las comunicaciones por lo que sea, se atascan. Puesto que el control de los motores se hace en velocidad, en principio el robot seguiría avanzando a la última velocidad comandada, una vez que se atascan las

comunicaciones. Adicionalmente, comprueba si se ha pulsado la tecla fin en el EyeBot, en caso afirmativo, finalizaría la ejecución del programa `emovil.c`.

```
void watchdog()
/*thread to detect END button pushed and to check watchdog */
{
 int counter=0;

 while((KEYRead()) != KEY4){
  counter=receivedmessages;
  OSSleep(50);
  /* watchdog: if no more messages received stops the motors */
  if (counter==receivedmessages)
  {
   if (!watchdog_on)
    {LCDPrintf("watchdog\n");
     watchdog_on=1;}
   drive=0.;
   steer=0.;
  }
 }
 OSKill(slave_p[1]);
 OSKill(slave_p[2]);
 OSKill(0);
}
```

Capítulo 5

Conclusiones y Mejoras

Tras haber explicado en capítulos anteriores en qué consiste y como hemos implementado nuestro comportamiento, terminamos ahora este proyecto comentando las conclusiones obtenidas, las dificultades principales que hemos identificado y las posibles mejoras para un funcionamiento más eficiente.

5.1. Conclusiones

En primer lugar resaltar que hemos conseguido el objetivo principal de este proyecto, que el robot siga a la pelota en tiempo real mediante el uso de una cámara cenital como sensor principal del comportamiento. Para ello, hemos hecho uso de un PC para el análisis de la imagen, de un módulo de radio para comunicar el PC con el robot, y obviamente de un robot EyeBot que se mueve por el entorno.

El diseño ha sido concebido siguiendo la arquitectura JDE [Cañas02]. Para ello hemos dividido nuestro comportamiento en dos esquemas que se ejecutan simultáneamente. Uno de ellos es el esquema perceptivo, consistente en analizar la imagen; y el otro es un esquema motriz, encargado de crear y enviar las órdenes de control apropiadas para el movimiento de lo motores, tal y como vimos en el capítulo cuatro.

Hemos implementado el proyecto en dos programas, el `ebase.c`, que corre en el PC y el `emovil.c` que se ejecuta en el Eyebot. En `ebase.c` se analizan las imágenes capturadas por la cámara cenital, con la finalidad de hallar en ellas el ángulo y la distancia relativas entre el EyeBot y la pelota. Con estos datos se toman unas decisiones de movimiento que se materializan en órdenes que se mandan al EyeBot estableciendo para ello una comunicación por radio. El `emovil.c` recibe este mensaje y mueve los motores conforme a la orden recibida. Para ello, dispone de tres hebras, una primera receptora encargada de recibir y analizar el mensaje proveniente del PC; otra encargada de mover los motores de acorde al mensaje, y una última que espera a que se pulse una tecla para la finalización del programa y sirve de mecanismo de “perro de guarda”

Se ha conseguido analizar la imagen a un ritmo superior de 7 imágenes por segundo. Con ello, hemos superado uno de los requisitos implícitos que teníamos, que el análisis de la imagen no fuera a un ritmo inferior de 4 imágenes por segundo. De este modo, hemos conseguido que el EyeBot siga a la pelota en tiempo real, y se ha mejorando la suavidad de movimiento obtenida en un proyecto anterior, el sigue pelota con visión local [SanMartín02]. En éste solo se consiguió analizar la imagen a un ritmo de 3 imágenes por segundo.

El filtro realizado para distinguir al EyeBot y a la pelota ha sido un filtro de color basado en el espacio HSI. No lo hemos realizado en el espacio RGB, porque el HSI es más robusto que éste ante cambios en la luminosidad. Un problema que hemos tenido, es que, aunque hemos comprobado que es más robusto el filtro de color en el espacio HSI que el RGB, sólo responde bien ante cambios ligeros en la luminosidad (sombras, más o menos intensidad, ...). Es decir, hemos podido ver que, cambiando de la luz natural a la luz artificial, el mismo filtro no respondía correctamente a ambas situaciones, teniendo que cambiar los valores del rango del Matiz (H) y de la Saturación (S).

Para poder establecer una comunicación fiable entre el PC y el EyeBot, dentro del `ebase.c` hemos creado dos procesos que se comunican a través de un pipe. El proceso Padre, que sirve de sincronizador, escribe en la entrada del pipe un carácter cada 100 milisegundos, para avisar al proceso Hijo que puede transmitir por radio. De este modo se ha conseguido un envío contínuo de mensajes (*streaming*) entre el PC y el EyeBot. Si hubiéramos enviado los mensajes a un ritmo superior, se hubiera saturado la capacidad del búffer, en contra, a un ritmo más lento, hubiera provocado una disminución en la calidad del movimiento.

Otra de las dificultades que hemos identificado es que no hemos podido separar las dos partes de las que se compone el `ebase.c` (el análisis de la imagen y la toma de decisión de control) en dos hebras. La causa es que éstas se ejecutan dentro del proceso donde se inicializa la radio, y la librería de radio del PC dificulta el control del ritmo de emisión, ya que captura la señal `SIGALARM`. Por ello no pudimos usar hebras asíncronas, pues no había manera de controlar el ritmo de sus iteraciones de modo eficiente: `usleep` no funciona bien con `SIGALARM` capturada. Esto viola la correspondencia uno a uno entre los esquemas y las hebras que los implementan. Con esta implementación percepción y actuación están acoplados, no obstante los resultados prácticos han sido satisfactorios.

Cabe señalar que los resultados de este proyecto, el código fuente de los programas

desarrollados, y un vídeo demostración, están disponibles en la web del grupo [1].

5.2. Líneas Futuras

Una de las posibles mejoras de este proyecto es el empleo del filtro de Kalman en el seguimiento de la pelota. Con este filtro podríamos predecir hacia donde se dirige la pelota según la trayectoria que vaya efectuando. Debido a esto, podríamos ahorrar cálculos, ya que al predecir el movimiento de la pelota en la imagen, podremos ir limitando las zonas de la imagen a filtrar. También el robot alcanzaría más rápidamente a la pelota, ya que al saber hacia donde se dirige, se anticiparía yendo directamente hacia esa zona, y no persiguiendo permanentemente su posición actual.

Nuestro comportamiento sólo ha sido implementado para un sólo robot EyeBot, por lo que podríamos ampliarlo usando más jugadores. Deberíamos poder hallar la posición de todos ellos desde la imagen, y que el que esté situado más cerca de la pelota sea el que vaya hacia ella, esquivando a su vez a los adversarios que se encuentre en su camino.

Con el empleo de la cámara cenital, podremos desarrollar nuevos comportamientos dentro del marco de la RoboCup, como el poder pasar la pelota de un EyeBot a otro, o chutar a portería marcando gol.

[1] http://gsyc.escet.urjc.es/robotica

Bibliografía

[Garcia02] GARCÍA, E., CAÑAS, J. M., MATELLÁN, V.: *"Manual del Robot EyeBot"*, Informe Técnico del GSyc, 2003.

[Cañas03b] CAÑAS PLAZA, J. M.: *"Manual de visión en robótica"*, Informe Técnico del Gsyc, 2003.

[Cañas02] CAÑAS, J. M., MATELLÁN, V.: *"Dynamic schema hierarchies for an autonomous robot.*Actas de la VIII Conferencia Iberoamericana sobre Inteligencia Artificial(IBERAMIA 2002), Universidad de Sevilla, Noviembre 2002.

[García02] GARCÍA, E.: *"Construcción de un teleoperador para el robot EyeBot."*, Proyecto Fin de Carrera, Universidad Carlos III, 2002.

[SanMartín02] SAN MARTÍN, F.: *"Comportamiento sigue pelota en un robot con visión local."*, Proyecto Fin de Carrera, Universidad Rey Juan Carlos, 2002.

[Gomez02] GÓMEZ GÓMEZ, V. M.: *"Comportamiento sigue pared en un robot con visión local."*, Proyecto Fin de Carrera, Universidad Rey Juan Carlos, 2002.

[Crespo03] CRESPO DUEÑAS, M. A.: *"Localización probabilística en un robot con visión local."*, Proyecto Fin de Carrera, Universidad Politécnica, 2003.

[Matute03] MATUTE BAENA, A.: *"Filtro de color configurable."*, Proyecto Fin de Carrera, Universidad Rey Juan Carlos, 2003.

[Lobato03] Lobato Bravo, D.: *"Evitación de Obstáculos basada en ventana dinámica."*, Proyecto Fin de Carrera, Universidad Rey Juan Carlos, 2003.

[Alvarez02] Álvarez Rey, J. M.: *"Implementación basada en lógica borrosa de jugadores para la RoboCup."*, Proyecto Fin de Carrera, Universidad Rey Juan Carlos, 2002.

[Aguero02] Agüero Durán, C.: *"Protocolo de Encaminamiento para Redes Adhoc."*, Proyecto Fin de Carrera, Universidad Rey Juan Carlos, 2002.

[Martinez03] Martínez Used, A.: *"Biblioteca fiable de comunicación inalámbrica para los EyeBots."*, Proyecto Fin de Carrera, Universidad Rey Juan Carlos, 2003.