UNIVERSIDAD REY JUAN CARLOS

**Ingeniería Técnica en Informática de Sistemas**

Escuela Superior de Ciencias Experimentales y Tecnología

Curso académico 2001-2002

**Proyecto Fin de Carrera**

# Comportamiento sigue pared en un robot con visión local.

**Tutor**: José M. Cañas Plaza

**Autor**: Víctor M. Gómez Gómez

Septiembre 2.002

*Dedicado a mi familia*

*y amigos.*

# Agradecimentos.

Quiero dar las gracias de manera especial al grupo de robótica de la URJC. En particular a José Mª Cañas y Vicente Matellán por la ayuda y conocimientos facilitados para poder llevar a cabo este proyecto.

# Índice general

**Resumen** **1**

**1. INTRODUCCIÓN.** **2**

1.1. ROBOCUP. . . . . 3

1.2. GRUPO DE ROBÓTICA DE LA URJC. . . . . 4

1.3. PROYECTO SIGUE PARED CON VISIÓN. . . . . 6

**2. OBJETIVOS.** **8**

2.1. OBJETIVOS . . . . . 8

2.2. REQUISITOS. . . . . 10

2.2.1. Robot EyeBot. . . . . 10

2.2.2. Vivacidad y Robustez. . . . . 11

2.2.3. Arquitectura JDE. . . . . 11

**3. EYEBOT.** **13**

3.1. DESCRIPCIÓN HARDWARE. . . . . 14

3.1.1. Sensores. . . . . 14

3.1.2. Actuadores. . . . . 15

3.1.3. Interacción. . . . . 16

3.1.4. Comunicaciones. . . . . 17

3.2. DESCRIPCIÓN SOFTWARE. . . . . 17

3.2.1. Acceso a los motores. . . . . 18

3.2.2. Acceso a la cámara. . . . . 21

3.2.3. Acceso a los servos. . . . . 23

3.2.4. Acceso al teclado. . . . . 24

3.2.5. Acceso al LCD. . . . . 24

3.2.6. Recursos de Multiprogramación. . . . . 26

3.3. COMPILACIÓN CRUZADA. . . . . 28

**4. DESCRIPCIÓN INFORMÁTICA. 30**
4.1. ESTRUCTURA GENERAL. . . . . 30
4.2. ESQUEMA PERCEPTIVO. . . . . 33
4.2.1. Filtro de color. . . . . 33
4.2.2. Establecer la frontera entre el suelo y las paredes. . . . . 36
4.2.3. Segmentación de la línea frontera. . . . . 39
4.3. ESQUEMA DE CONTROL. . . . . 44

**5. CONCLUSIONES Y MEJORAS 49**
5.1. CONCLUSIONES . . . . . 49
5.2. LINEAS FUTURAS . . . . . 51

**Bibliografía 53**

# Índice de figuras

1.1. Mascota AIBO de Sony. . . . . 3
1.2. Robot bípedo y robots de pequeño pequeño de la Robocup. . . . . 4
1.3. Robot Pioneer . . . . . 6

3.1. EyeBot de la URJC y distintas configuraciones del EyeBot. . . . . 13
3.2. Vista posterior y frontal de la cámara . . . . . 15
3.3. Display y botonera . . . . . 16
3.4. Esquema de descarga del código ejecutable . . . . . 28

4.1. Arquitectura del comportamiento. . . . . 31
4.2. Estructura de la implementación. . . . . 32
4.3. Imagen sin filtrar y filtrada del entorno . . . . . 36
4.4. Imagen filtrada e imagen con línea frontera . . . . . 39
4.5. Primera derivada ideal y real de la recta. . . . . 40
4.6. Proceso de segmentación. . . . . 43
4.7. Imagen filtrada e imagen con línea frontera segmentada . . . . . 43
4.8. Visualización de los segmentos en los casos todo pared y todo suelo. . . 45
4.9. Visualización de los segmentos en el caso de discontinuidad. . . . . 46
4.10. Visualización de los segmentos en el caso de pared infinita. . . . . 46
4.11. Visualización de los segmentos en el caso de esquina cóncava. . . . . 47
4.12. Visualización de los segmentos en el caso de esquina convexa. . . . . 48

# Índice de cuadros

3.1. Funciones de la interfaz VW . . . . . 20
3.2. Funciones de la interfaz VW (II) . . . . . 21
3.3. Funciones para la cámara . . . . . 22
3.4. Funciones para los servos . . . . . 23
3.5. Funciones para el teclado . . . . . 24
3.6. Funciones para la pantalla . . . . . 25
3.7. Funciones para las tareas en la multiprogramación. . . . . 27
3.8. Funciones para los semáforos . . . . . 28

# Resumen

El EyeBot es uno de los robots móviles sobre los que basa algunos de sus trabajos de investigación el grupo de la Universidad Rey Juan Carlos. Este robot dispone de una serie dispositivos, sensores y actuadores, que le permitirán desenvolverse por el entorno. Los sensores son la cámara, los encoders y los infrarrojos. Los actuadores son dos motores y dos servos. Además de estos dispositivos dispone de un microprocesador Motorola 68332 como elemento de proceso. El sistema operativo del robot incorpora una consola a través de la cual se pueden chequear los distintos componentes hardware y realizar la descarga de programas al robot. También ofrece una interfaz de programación para facilitar el acceso a los distintos dispositivos a través de los programas.

El proyecto de generar el comportamiento sigue pared mediante visión tiene como objetivo el que el robot EyeBot sea capaz de desplazarse con cierta velocidad paralelo a una pared, sirviéndose de la cámara como único elemento sensorial. De modo que deberá salir airoso de todas las posibles situaciones que se le presenten. Por ejemplo si se aleja de la pared que sigue deberá girar para acercarse, si se acerca a la pared deberá variar su trayectoria también para alejarse, si se acerca a una esquina cóncava tendrá que girar para evitar el choque con la pared frontal, etc. Para poder abordar estas situaciones el EyeBot deberá procesar las imágenes capturadas por su cámara para extraer la información relevante. Esta información sera la línea frontera entre el suelo y la pared que debe seguir. De manera que en función de como vea esta línea podrá calcular como esta orientado respecto a la pared y actuar en consecuencia.

Este proyecto ha sido desarrollado con una plataforma Linux, utilizando las herramientas que proporciona este sistema operativo, desde el editor para escribir los programas, hasta el procesador de documentos para la elaboración de la memoria, pasando por el compilador cruzado necesario para compilar los programas que se ejecutan en el EyeBot.

# Capítulo 1

# INTRODUCCIÓN.

A finales del siglo XX, principio de los años 70, surgió en el hombre la posibilidad de crear objetos artificiales que se desenvolvieran de modo autónomo, en gran medida gracias a los avances que se produjeron en aquella época en el campo de la microelectrónica. Estas serian las primeras incursiones serias en el denominado campo de la robótica. La robótica se define como el conjunto de conocimientos teóricos y prácticos que permiten concebir, realizar y automatizar sistemas basados en estructuras mecánicas poliarticuladas, dotados de un determinado grado de "inteligencia" y destinados a la producción industrial o a la sustitución del hombre en muy diversas tareas [1]. Desde esas incursiones de los años 70 hasta ahora el desarrollo en este campo ha experimentado un gran crecimiento, gracias sobre todo a los avances logrados en las últimas décadas.

Los primeros avances estuvieron enfocados principalmente a la utilización de robots en plantas industriales, para dotar de mayor flexibilidad a los procesos productivos. La mayoría de los robots utilizados en la industria son los denominados robots manipuladores, que estarían dentro de uno de los campos de estudio en los que se divide la robótica. Estos manipuladores están diseñados para mover piezas, herramientas, o dispositivos especiales, mediante movimientos variados. El caso de robots soldadores en plantas de montaje de automóviles, robots teleoperados en centrales nucleares, etc.

Otro de los campos de investigación y por el que más se ha preocupado últimamente el ser humano, es el campo de la robótica móvil. Que responde a la necesidad del ser humano de extender el campo de aplicación de la robótica, restringido inicialmente a una estructura mecánica anclada en sus extremos, el caso de los brazo robóticos. De este modo dentro de este área de la robótica móvil también se distinguirán entre dos

[1] http://www.geocities.com/Eureka/Office/4595/robotica.html

tipos de robots móviles. Los robots móviles teleoperados, es decir con acciones dirigidas a distancia por el ser humano, el caso de los robots desactivadores de explosivos, robots para la exploración de fondos marinos, etc. El otro tipo de robots móviles son los que actúan de manera autónoma, en los que la toma de decisiones a la hora de llevar a cabo una actuación no es supervisada por humanos, robots bibliotecarios, robots guias de museos, etc.

Pero no sólo se investiga en el campo de la robótica móvil para crear máquinas que trabajen, también se orienta hacia el ocio [Matellán00]. El caso de los robots mascota que están revolucionando la interacción humano-robot, como el robot de Sony llamado AIBO, que aparece en la figura 1.1, cuyo comportamiento evoluciona a medida que el dueño interactúa con él. O la realización de encuentros internacionales sobre robótica, en los que se realizan competiciones de diversos tipos, sumo, fútbol, etc., entre robots.

Figura 1.1: Mascota AIBO de Sony.

## 1.1. ROBOCUP.

Basado en esta rama de investigación, de la robótica como instrumento para el entretenimiento y la educación, se encuentra el campeonato de mundial de fútbol para robots, la llamada Robocup [2]. Esta competición nació en 1996 con la intención de incentivar el interés por la ciencia y la tecnología. Presenta un entorno dinámico, de tiempo real y distribuido donde se pueden investigar distintas tecnologías en generación de comportamiento autónomo. Por ello supone un escenario desafiante para la investigación en robótica, agentes, inteligencia artificial, etc. con el aliciente de la competición y la vistosidad. Cada año se celebra este evento que enfrenta a equipos de distintas

[2] http://www.Robocup.org

nacionalidades los cuales pueden competir en alguna de las diferentes categorías que existen en esta Robocup. Categorías como la que enfrenta a equipos de robots móviles de pequeño tamaño como los de la figura 1.2, en la compiten equipos formados por cinco robots cada uno, que compiten sobre un tablero de ping-pong con una pelota de golf de color naranja y ayudados por una cámara cenital. Cuyo procesamiento de imágenes lo realiza un PC, para determinar la posición y la actuación que deberán llevar a cabo los robots en función de esa posición. Lo que hace que el robot solo se ocupe de actuar, mientras la toma de decisiones es responsabilidad del PC.

Existen otras categorías que van desde la que enfrenta a equipos implementados en un simulador. Hasta llegar a categorías como la que enfrentan a humanoides o robots bípedos (figura 1.2), robots que intentan reproducir total o parcialmente la forma y el comportamiento cinemático del ser humano.

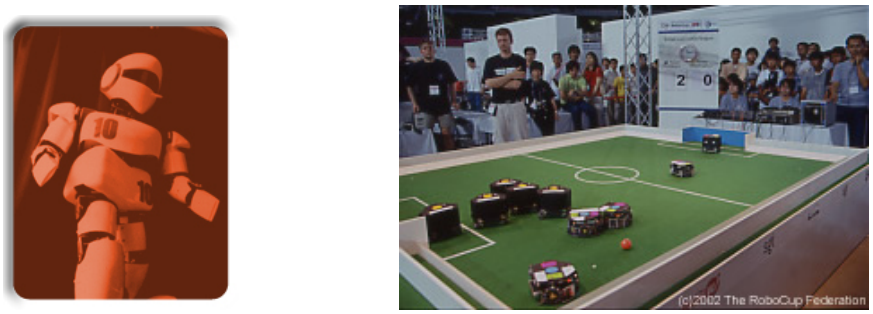


Figura 1.2: Robot bípedo y robots de pequeño pequeño de la Robocup.

## 1.2. GRUPO DE ROBÓTICA DE LA URJC.

El grupo de robótica de la Universidad Rey Juan Carlos [3] empezó a interesarse por esta competición en el año 2000, con idea de participar en un futuro no muy lejano. Con este motivo en primer lugar tuvieron que decidir si la participación sería con robots propios o por contra se comprarían unos ya construidos. El grupo contaba con robots del tipo *Lego Mindstorms* que se emplean para impartir la asignatura de Robótica. El inconveniente de utilizar estos robots móviles era que poseen poca capacidad de proceso y sus sensores no son muy efectivos. De este modo se decidió comprar los robots con los que se participaría en la Robocup. Los requisitos principales del robot que se buscaba, eran que fuera económicamente asequible, de un tamaño reducido para participar en la categoría de robots de pequeño tamaño de la Robocup, para la cual el robot empleado

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

debe caber en una circunferencia de 18 cm. de diámetro. Con gran capacidad de proceso para que fuera rápido en la toma de decisiones y procesando datos. De este modo tras evaluar las diferentes características de los robots móviles existentes en el mercado se decidió adquirir los robots EyeBot. Que estaban especialmente diseñados para la participación en la Robocup y eran los que más se asemejaban a las características deseadas, y del que se hará una descripción detallada más adelante en el capítulo 3.

Con las miras puestas en esta competición el grupo de robótica de la Universidad Rey Juan Carlos empezó a desarrollar una serie de proyectos enfocados en la propia competición como el simulador basado en lógica borrosa para jugadores de la Robocup [Álvarez02], o el seguimiento de jugadores para la Robocup basado en color aun en desarrollo. Y por otra parte otros trabajos que tenían como principal protagonista al robot móvil EyeBot, el caso del proyecto que consistía en la elaboración de un teleoperador para dirigir el EyeBot desde el PC [GarcíaMorata02], el primero que se realizo sobre el EyeBot. Cuyo principal objetivo era el conocimiento a fondo de las posibilidades que ofrecía este robot móvil. O la creación de un protocolo de comunicaciones para el EyeBot, [Agüero02].

Además de estas líneas de investigación, el grupo centra algunos de sus trabajos en el desarrollo de comportamientos para robots de interiores, cuya arquitectura de control esta basada en la arquitectura de control JDE (*Jerarquía Dinámica de Esquemas*) para robots autónomos [Cañas02b]. Esta arquitectura JDE basa el comportamiento, que debe desarrollar el robot autónomo, en una jerarquía de denominados esquemas, que se encargan de realizar una labor específica. De este modo que en cada momento se ejecuta una de ellas dependiendo de la prioridad asignada a cada una. Estos esquemas básicamente se encargan de la tarea perceptiva unos y de elaborar estímulos complejos otros. Los perceptivas tienen como objetivo recabar información relevante del entorno para saber en cada momento la situación del robot. Mientras que los de control tienen como misión establecer las ordenes de control sobre el robot en cada momento en función de la información obtenida por las perceptivas. Dentro de esta línea de trabajo estaría incluido el desarrollo del comportamiento sigue pared mediante visión para el robot EyeBot. Además de otros trabajos de investigación para entornos interiores que se llevan a cabo y que tienen como protagonista al robot Pioneer (figura 1.3). El Pioneer es un robot de tamaño mediano para entornos de interiores. Sensorialmente

está dotado de una corona de 16 ultrasonidos, un cinturón de sensores táctiles y un encoder 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, permite aplicaciones como construcción de mapas,navegación, etc. El cómputo necesario viene 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.

Figura 1.3: Robot Pioneer

## 1.3. PROYECTO SIGUE PARED CON VISIÓN.

Dentro de estos proyectos desarrollados por el grupo de robótica de la URJC, enfocados a la investigación y desarrollo de comportamientos para robots móviles en entornos interiores, se encuentra el proyecto de generar el comportamiento sigue pared lateral con visión local para el robot EyeBot. Que además sirve también para explorar un poco más las posibilidades de este robot, con idea de participar con él en la Robocup. Este proyecto es uno de los primeros que desarrolla un comportamiento totalmente autónomo para el robot EyeBot, de todos los proyectos desarrollados hasta el momento para este robot, junto con el proyecto sigue pelota mediante visión local [SanMartín02]. El objetivo que se desea conseguir desarrollando este comportamiento, es que el robot móvil sea capaz de seguir una pared lateral a él mediante la visión a través de su cámara, evitando colisiones haciendo rectificaciones en su trayectoria cuando haya una pared frontal, girando cuando se encuentre una esquina convexa, etc.

El desarrollo de comportamiento sigue un esquema de implementación que se englobaría dentro de la denominada arquitectura JDE para el control de robots autónomos, en la cual basa sus trabajos el grupo de Robótica de la URJC.

La presente memoria detalla todos los aspectos relevantes a la hora de llevar a cabo el desarrollo del comportamiento sigue pared mediante visión en un robot EyeBot. Esta memoria se articula en cinco capítulos. El capítulo 2 describe los objetivos y requisitos planteados a la hora de desarrollar el comportamiento. El capítulo 3 hace una descripción detallada del robot EyeBot que será el empleado para generar el comportamiento, tanto a nivel software como hardware haciendo hincapié en los recursos utilizados en este proyecto. La descripción informática o implementación será detallada en el capítulo 4. Para concluir, en el capitulo 5, con la presentación de las principales conclusiones extraídas, así como las posibles líneas de mejora por las que se puede continuar el trabajo realizado en este proyecto.

# Capítulo 2

# OBJETIVOS.

En este capítulo se van a describir los objetivos que se quieren conseguir con la generación del comportamiento sigue pared con visión. Así como los requisitos que debe cumplir el desarrollo o implementación de este comportamiento y el propio comportamiento, con los que debe ser capaz el robot de abordar todas las posibles situaciones a la hora de desenvolverse por el entorno para el que esta pensado este comportamiento.

El proyecto de generar el comportamiento sigue pared mediante visión local tiene como principal objetivo el que el robot EyeBot sea capaz de desplazarse paralelo a una pared lateral, a cierta distancia de esta pared y con cierta velocidad. Para ello usará la cámara como único elemento sensorial, a través de la cual extraerá información del entorno. De modo que deberá ser capaz de actuar de la manera apropiada en cada una de las posibles situaciones que se le presenten. Por ejemplo si se aleja de la pared lateral que sigue deberá girar hacia esta pared para acercarse, si se acerca a la pared deberá variar su trayectoria también pero esta vez para alejarse con el único propósito de mantener cierta distancia con la pared a seguir, si se acerca a una esquina cóncava tendrá que girar para evitar el choque con la pared frontal, etc. Para poder abordar estas situaciones el EyeBot deberá procesar las imágenes capturadas por su cámara para extraer la información relevante. Esta información será la línea frontera entre el suelo y la pared que debe seguir. De manera que en función de como vea esta línea podrá calcular su posición y orientación respecto a la pared, y actuar en consecuencia.

## 2.1. OBJETIVOS

El principal objetivo del diseño será identificar todas las situaciones o casos relevantes que se encontrara el EyeBot y en los que deberá ser capaz de actuar de modo

adecuado, para poder desarrollar el comportamiento sigue pared mediante visión.

Para llevar a cabo este objetivo de desarrollar este comportamiento para el robot EyeBot, basaremos la estructura de este en la arquitectura de control JDE con la que se trabaja en el grupo de Robótica de la URJC. La arquitectura JDE mencionada en el apartado 1 basa el comportamiento, que debe desarrollar el robot autónomo, en una jerarquía de denominados esquemas, que se encargan de realizar una labor específica. De este modo que en cada momento se ejecuta una de ellas dependiendo de la prioridad asignada a cada una. Estos esquemas básicamente se encargan de la labor perceptiva unos y de elaborar estímulos complejos otros, la actuación. Los perceptivas tendrán como objetivo recabar información del entorno para saber en cada momento la situación del robot. Esta información será la línea frontera entre el suelo y la pared. Mientras que los de control tienen como misión establecer las ordenes de control sobre el robot en cada momento en función de la información obtenida por las perceptivas. El esquema perceptivo del comportamiento deberá ser capaz de extraer la línea frontera de las imágenes capturadas por el robot. El esquema de control por su parte deberá analizar todos los casos relevantes ante los que se encuentre el robot para indicar a este la actuación adecuada en cada momento.

De este modo debe estudiarse la viabilidad de aplicar esta arquitectura de control al desarrollo del comportamiento sigue pared mediante visión para el EyeBot, atendiendo a las características de este robot, tanto a nivel hardware como a nivel software.

La implementación que se lleve a cabo del comportamiento sigue pared mediante visión en un robot EyeBot debe ser capaz de validar el diseño especificado anteriormente. De modo que este implementación también se dividirá en dos partes como ocurría con la arquitectura de diseño. Una parte de la implementación será la que lleve a cabo el esquema perceptivo de la arquitectura de diseño, mientras que la otra parte se encargara de la actuación del robot EyeBot.

El objetivo de la parte de la implementación encargada de la percepción deberá ser capaz de identificar los estímulos relevantes de las imágenes capturadas por el EyeBot. Para poder seguir la pared debe reconocer en las imágenes la línea frontera entre el suelo y la pared. Esta información será extraída de las imágenes realizando sobre estas distintas técnicas de procesado de imágenes. El objetivo de estas técnicas será diferenciar en las imágenes el suelo de las paredes, de modo que quede definida en las imágenes

esa línea frontera entre el suelo y las paredes, para llevar a cabo esto se buscara aplicar un filtro de color en las imágenes de modo que se reconozca en estas el color del suelo discriminando este de las paredes. Pero además se deberá seguir aplicando técnicas de tratamiento de imágenes para definir del un modo más preciso esta línea frontera. Para ello tras el filtro de color será necesario determinar con exactitud esta línea, es decir que puntos la componen de manera que será necesario implementar una función que determine los puntos de esta frontera. Para concluir el procesado de las imágenes realizando una segmentación de estas, para determinar los segmentos que componen la línea frontera en cada una de las imágenes capturadas. De modo que tengamos totalmente caracterizada la línea frontera entre el suelo y la pared.

Además de desarrollar distintas técnicas de procesado y tratamiento de imágenes, será necesario estudiar técnicas de control, para realizar la implementación del esquema de control encargado de la actuación del robot. Técnicas que nos permitan mover adecuadamente al robot EyeBot en función de lo percibido.

## 2.2. REQUISITOS.

A la hora de generar el comportamiento sigue pared mediante visión por parte del EyeBot, a través de la consecución de los objetivos descritos anteriormente. Hay que tener en cuenta que se deben cumplir una serie de requisitos, para que el comportamiento sea lo más completo y eficiente posible. Estos requisitos serán, unos desde el punto de vista del diseño y la implementación, como generar el comportamiento basándonos en la arquitectura JDE, y otros desde el punto de vista de eficiencia y robustez, dentro de estos se encuadraría el que sea un comportamiento vivaz, es decir que sea rápido en la percepción para actuar rápido en el control para así evitar colisiones, etc. Y por último el que se desarrolle en condiciones luminosidad poco variables para que sea mas robusto en cuanto a la percepción.

### 2.2.1. Robot EyeBot.

El principal requisito que se debe cumplir al generar este comportamiento sigue pared mediante visión, es que debe ser implementado para su desarrollo en el robot EyeBot. De modo que debemos ser capaces de realizar esta tarea sirviéndonos de los elementos y facilidades de las que dispone este robot, que se describirán en el apartado

3.

### 2.2.2. Vivacidad y Robustez.

Cuando el robot EyeBot este llevando cabo el comportamiento, lo más deseable es que de impresión de dinamismo, que sea lo más reactivo posible. Es decir que sea rápido en la toma de decisiones y este el menor tiempo posible procesando información y deliberando, y la mayoría del tiempo actuando.

De este modo se desarrollara en tiempo real, porque si le llevara demasiado tiempo procesar la información percibida por la cámara, la actuación no se correspondería en el tiempo con la información percibida en cada momento, si no que actuaria con retardo.

Por lo tanto este requisito de desarrollar el comportamiento en tiempo real deberá estar presente cuando se vaya a implementar el comportamiento, y más concretamente cuando se intenten llevar a cabo los objetivos perceptivos descritos anteriormente.

A la hora de generar el comportamiento sigue pared mediante visión para el robot EyeBot, se buscara que este sea lo más completo y robusto posible. Es decir que pueda llevarse a cabo en distintos entornos, con condiciones físicas distintas, como la luminosidad, niveles de batería, etc. Por lo tanto otro requisito que se debe tener en cuenta es que el comportamiento sea lo más versátil posible, de tal modo que a la hora de intentar conseguir los objetivos perceptivos no le influyan las condiciones de luz a la hora de percibir los estímulos del entorno,a través de la cámara. Y por otro lado que no le influya el trabajar con niveles de batería distintos cuando deba llevar a cabo la actuación.

### 2.2.3. Arquitectura JDE.

La arquitectura de control para robots autónomos en la que basa el desarrollo de comportamientos el grupo de robótica de la URJC, es la arquitectura JDE. De tal modo que este requisito está impuesto a priori, incluso antes de analizar los objetivos que se buscaran al desarrollar el comportamiento.

De esta manera este requisito no sólo se tendrá en cuenta a la hora de llevar a cabo la implementación. Si no que a la hora de analizar los objetivos que se desean alcanzar al generar el comportamiento, se tuvo en cuenta que la estructura general de este se basaría en esta arquitectura de control. Esta arquitectura, va a dividir el

comportamiento a desarrollar en un esquema perceptivo por un lado, y otro de control. Estos esquemas interactuaran entre si para generar el comportamiento.

# Capítulo 3

# EYEBOT.

El robot EyeBot es uno de los robots móviles con los que cuenta en el grupo de robótica de la URJC para la realización de proyectos y trabajos. El robot móvil EyeBot fue adquirido por el grupo de robótica, con vistas a la participación en próximas ediciones del campeonato mundial de fútbol para robots, la Robocup.

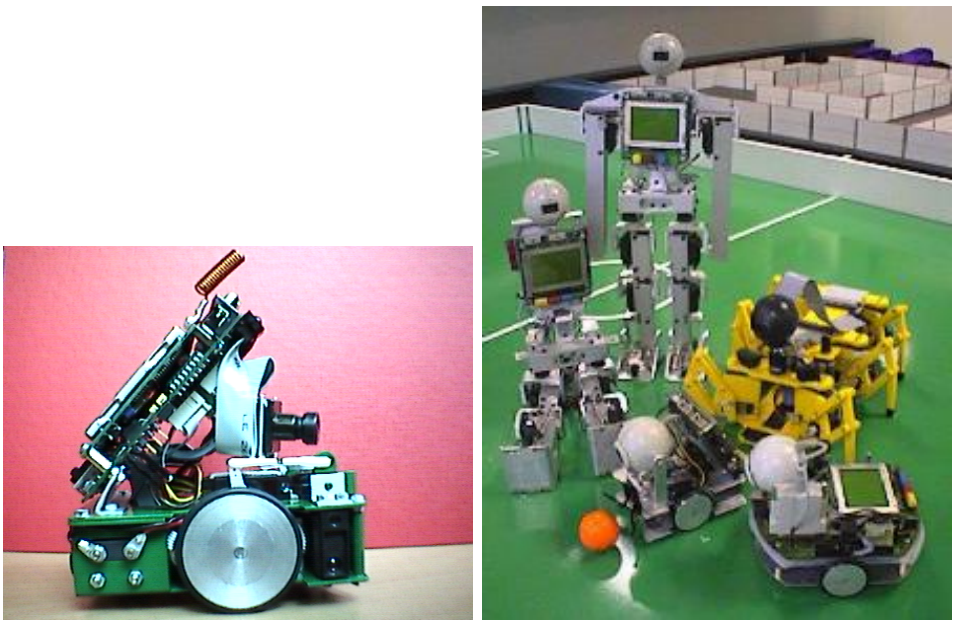

Figura 3.1: EyeBot de la URJC y distintas configuraciones del EyeBot.

En este capítulo describiremos los distintos elementos hardware que componen el EyeBot del grupo de robótica de la URJC, así como las funciones que proporciona el sistema operativo para acceder a ellos, haciendo hincapié en los actuadores y sensores que utilizamos en este proyecto.

## 3.1. DESCRIPCIÓN HARDWARE.

A nivel hardware el EyeBot dispone de una placa base a la esta conectada un microprocesador, al que a su vez van conectados unos dispositivos u otros según la configuración elegida. El microprocesador es un Motorola 68332 a 35 MHz con una memoria Flash-ROM de 512 KB que son para el sistema operativo. Para los programas de usuario dispone de una memoria RAM de 1 MB que permite ejecutar los programas almacenados en la ROM. Dispone de 2 puertos serie, 1 paralelo, además tiene 8 entradas digitales, 8 salidas digitales y 8 entradas analógicas.

Se trata de un robot móvil que puede tener distintas configuraciones (figura 3.1) a nivel físico en función de la utilidad a la que se vaya a destinar. La configuración con la que se trabaja en la URJC esta especialmente orientada a la participación en la Robocup (figura 3.1).

El EyeBot utilizado por el grupo de robótica de la URJC tiene incorporados distintos elementos, a nivel físico, que le van a permitir ser capaz de realizar distintas tareas: sensores, actuadores, elementos de interacción y de comunicaciones.

### 3.1.1. Sensores.

Dispone de varios sensores a través de los cuales recoge información del entorno para posteriormente tratarla y desenvolverse por él. Estos sensores se pueden dividir en tres grupos dependiendo de su naturaleza.

Por un lado los tres sensores infrarrojos, que se encargan de medir la distancia a un obstáculo cercano a través de la cantidad de luz rebotada tras su emisión.

Otros sensores de los que dispone son los dos encoders o cuenta vueltas, situados en cada motor y que devuelven un número de pulsos, mediante los cuales se puede comprobar cuanto se desplazó cada rueda y en que dirección.

Por último la cámara, el elemento principal y único de percepción para el desarrollo del comportamiento sigue pared con visión. La cámara se sitúa en el centro de la parte frontal del EyeBot. Trabaja con 24 bits en color o en escala de grises proporcionando una resolución de 80 x 60 pixels lo que hace que su procesamiento pueda ser relativamente rápido. La velocidad máxima de captura de imágenes es de 3.7 imágenes por segundo aproximadamente. El enfoque de la cámara no es automático, sino que se realiza girando el objetivo hasta encontrar el enfoque adecuado. Ofrece la posibilidad de trabajar con ella en auto-brillo o no, esta opción se configura a través del programa.

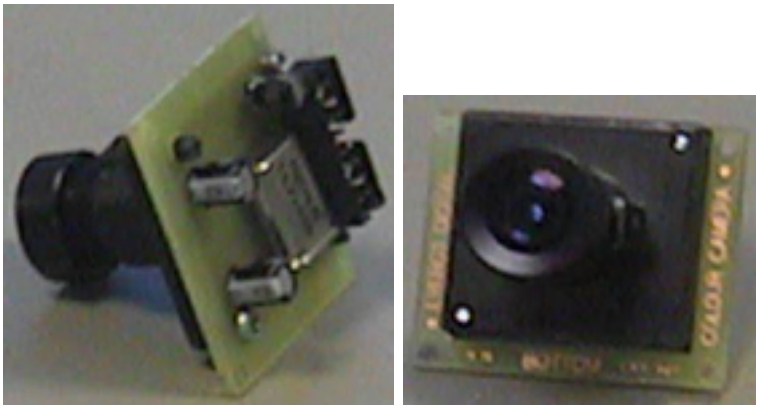

Figura 3.2: Vista posterior y frontal de la cámara

La cámara ha sido elaborada con la última tecnología CMOS que proporciona valores de brillo mejores que las convencionales basadas en tecnología CCD, y que permite hacer un ajuste automático de este. El tamaño de la cámara es de 3.0 cm x 3.4 cm.

### 3.1.2. Actuadores.

Se puede diferenciar entre dos tipos de actuadores, unos le sirven como medio de locomoción como es el caso de los dos motores de continua, y otros cumplen una función mas específica, como es el caso de los servos.

Los dos motores de continua son los responsables de la movilidad del EyeBot fijándo su velocidad, siendo los valores negativos los responsables del desplazamiento hacia atrás, los positivos hacia delante, y con el cero permanecen parados. Estos motores hacen que las ruedas motrices del EyeBot no cambien de dirección, para efectuar giros uno de los dos motores girara más despacio que el otro, haciendo que el EyeBot tenga movimientos tipo tanque.

Por su parte los dos servos, que son motores controlados por pulsos PWM. A diferencia de los motores de continua estos pueden ser posicionados y enclavados en un ángulo determinado. Estos servos se encargan de realizar tareas específicas, uno es el que permite orientar la cámara con distintos ángulos, ya sea en horizontal o vertical dependiendo del modelo de EyeBot. El otro servo es el encargado del "pateador"que es que se utilizara para golpear la bola cuando participe en la Robocup.

El servo encargado de la orientación de la cámara será otro elemento importante en la elaboración del comportamiento para el robot. Esto es debido a que necesitamos que

el robot se mueva en paralelo a la pared que va a seguir, por lo tanto para no perder esta la cámara deberá estar orientada hacia la pared que va a seguir, que será lateral. Para poder obtener un mejor ángulo de visión con la cámara, el ángulo del servo que orienta a esta será el máximo posible hacia el lado donde esta la pared a seguir.

### 3.1.3. Interacción.

Los elementos de interacción que tiene el robot EyeBot son: los altavoces, el micrófono, la botonera y la pantalla LCD. Van a permitir al usuario interactuar con el sistema.

La pantalla LCD se encuentra en la parte posterior del EyeBot, y consta de una rejilla de 128 x 64 pixels, desde el que se pueden visualizar hasta 17 caracteres ASCII por línea, con un total de 8 líneas (la línea inferior está reservada para las etiquetas de menú).

Será un elemento importante en el desarrollo del comportamiento sigue pared, ya que a través del LCD podemos ver las imágenes que esta viendo el EyeBot en cada momento, así como otro tipo de información que nos ayudara a seguir el comportamiento del robot.

La botonera se encuentra en la parte inmediatamente inferior a la pantalla LCD. Hay una hilera de 4 botones cuya funcionalidad dependerá del programa o actividad que se esté ejecutando en cada momento. Será utilizada en el desarrollo del proyecto, para poder elegir entre distintos tipos de visualización en el LCD de las imágenes recogidas por el robot. Así como para indicar el fin de la fase de calibración y para indicar el fin de la ejecución del programa cuando lo consideremos oportuno.


Figura 3.3: Display y botonera

### 3.1.4. Comunicaciones.

Permiten al EyeBot interactuar con otros elementos externos a él, como con otros EyeBot o con el PC. Los elementos de comunicaciones que tiene el EyeBot son: el puerto serie, el puerto paralelo y la radio.

El puerto serie utiliza un conector RS-232 situado en la parte frontal del robot es posible conectar el EyeBot a un PC. De este modo se posibilita la descarga de programas o el envío de comandos, datos y medidas entre distintas plataformas. Estas transmisiones se pueden realizar a diferentes velocidades, 9600, 19200, 38400, 57600 y 115200 baudios.

Por su parte el puerto paralelo se encuentra al lado del puerto serie. Se utiliza para conectar el EyeBot con el ordenador y poder realizar una depuración del programa con las herramientas que suministra Motorola para ello.

Por último el módulo de radio permite la comunicación inalámbrica entre robots sin la necesidad de un ordenador central, también permite la comunicación sin cables entre robot y PC. El módulo de radio utiliza uno de los puertos serie del robot. Esta red trabaja con token ring virtual (paso de testigo). Características técnicas: Frecuencia de trabajo a 433MHz, velocidad máxima de transmisión de 9600 baudios, se incluye un protocolo de tolerancia a fallos, transmisión de 8 bits.

## 3.2. DESCRIPCIÓN SOFTWARE.

Desde el punto de vista software el EyeBot dispone de un sistema operativo propio denominado *Robios* (Robot Basic I/O System). Consta de una consola (*shell interactiva*), una tabla de dispositivos hardware conectados (la *HDT*) y una interfaz de programación (*API*) que permite al usuario el acceso a los distintos elementos del sistema de manera sencilla.

A través de la consola el sistema operativo ofrece la posibilidad de cargar los programas de usuario, y de ejecutarlos. También es posible ejecutar distintos programas de demostración del funcionamiento del sistema.

La interacción de la consola con el usuario se realiza mediante menús en los que el usuario va eligiendo opciones en pantallas pulsando botones.

A través de la interfaz de programación se puede acceder a encoders, motores, cámara, etc. Es la interfaz que utilizaran los programas que materializan el comportamiento sigue pared.

Además de esta interfaz de programación del sistema operativo hay otras librerías proporcionadas por el fabricante que también utilizaremos.

### 3.2.1. Acceso a los motores.

Estas funciones de acceso a los motores nos permitirán poner en movimiento al EyeBot (cuadros 3.1 y 3.2).

Existen dos maneras de acceder a estos motores, el acceso directo a los mismos (control en lazo abierto) o mediante la librería de control VW (control en bucle cerrado con realimentación). La manera más aconsejable de acceder a los motores es con el control VW.

Con el acceso directo a los motores es necesaria una inicialización previa utilizando la función "MOTORInit(nombre_motor_en_HDT)". Tras esto se puede indicar al motor deseado que se desplace con la función "MOTORDrive(motor,velocidad)", fijando la velocidad en porcentajes con valores desde -100 a 100. La velocidad será positiva si se desea un desplazamiento hacia delante y negativa si se desea hacia atrás.

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

En el caso particular del comportamiento sigue pared se ha hecho uso del control VW. El uso de este tipo de control requiere la inicialización del mismo, esto se realiza a través de la función "VWInit(..)" con la que se obtiene un manejador para poder hacer uso del resto de funciones del control VW. Posteriormente se debe poner en funcionamiento un controlador que regula la energía suministrada a los motores mediante la función "VWStartControl(..)", a la que se le pasan como parámetros, además del manejador obtenido anteriormente, cuatro números reales que representan las componentes proporcionales e integrales de velocidad lineal y angular.

Con la librería VW, de acceso a los motores del EyeBot, se pueden realizar los movimientos de dos maneras distintas: con control en velocidad o con control en posición.

Para los movimientos con control en velocidad se emplea la función "VWSetSpeed(..)" a la que solo se pasa como parámetros la velocidad lineal y angular a la que queremos que se mueva el robot, para detener el robot se le pasara velocidad lineal y angular cero.

Para los movimientos con control en posición, se debe pueden usar una serie de funciones como "VWDiveStraight(..)" para avanzar recto a la que se le pasa la velocidad lineal a la que queremos que avance y la distancia en metros a avanzar. "VWDriveTurn(..)" para realizar movimientos angulares sin avanzar que tiene como parámetros el ángulo que queremos que gire y la velocidad angular con que realizara el giro. "VWDriveCurve(..)" que permite avanzar describiendo una trayectoria curvilínea pasando a la función cierta velocidad angular y se le pasara la longitud de la trayectoria a realizar, el ángulo con el que la realizara en radianes, y la velocidad lineal que llevara en m/s. Además a todas estas funciones se les pasa también como parámetro el manejador para acceder a los motores. Con el control de movimientos en posición el EyeBot avanzara a una determinada velocidad la distancia o el ángulo que le indiquemos, en metros o radianes, y luego parara.

Los tipos de movimiento que debe llevar a cabo el robot para conseguir desarrollar el comportamiento son movimientos lineales y movimientos curvilíneos. Estos se pueden realizar con el uso de la función, de control en velocidad de la librería VW.

Cuando dejemos de trabajar con los motores de continua, a través de la librería de control VW, será necesario deshabilitar el controlador para el acceso a los motores. Esto se realiza usando la función"VWStopControl(..)", a la que se le pasa como parámetro el manejador obtenido al principio a través de la función"VWInit(..), para acceder a los motores.

| *Formato Función* | *P.Entrada* | *P.Salida* | *Descripción* |
|---|---|---|---|
| Interfaz VW | | | |
| VWHandle VWInit (DeviceSemantics semantics, int Timescale) | (semantics) Nombre del V-Omega Driving<br>(Timescale) Escala de actualización (1 to ..) | Manejador del V-Omega Driving<br>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<br>-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<br>(v) velocidad lineal<br>(w) velocidad de rotación | 0=ok<br>-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<br>(vw) Puntero a una estructura que almacena los valores actuales de la velocidad lineal y angular | 0=ok<br>-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<br>(x) posición sobre el eje x en metros<br>(y) posición sobre el eje y en metros<br>(phi) Orientación en radianes | 0=ok<br>-1= manejador incorrecto | Fija la posición del robot. |
| int VWGetPosition(VWHandle handle, PositionType* pos) | (handle) Manejador VW-Driver<br>(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<br>(Vv) Parámetro para la componente proporcional del controlador de velocidad lineal<br>(Tv) Parámetro para la componente integral del controlador de velocidad lineal<br>(Vw) Parámetro para la componente proporcional del controlador de velocidad angular<br>(Tw) Parámetro para la componente integral del controlador de velocidad angular | 0=ok<br>-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<br>-1= manejador incorrecto | Deshabilita el controlador (PI-Controller). |
| int VWDriveWait (VWHandle handle) | (handle) Manejador VW-Driver. | -1= Manejador incorrecto.<br>0 = El comando previo VWDrivex ha sido completado. | Bloquea la llamada a procesos hasta que el comando previo VWDrivex haya sido completado |

Cuadro 3.1: Funciones de la interfaz VW

| *Formato Función* | *P.Entrada* | *P.Salida* | *Descripción* |
|---|---|---|---|
| int VWDriveStraight (VWHandle handle, meter delta, meterpersec v) | (handle) Manejador VW-Driver<br>(delta) distancia a conducir en m (positiva adelante) (negativa atrás)<br>(v) velocidad (siempre positiva). | 0 = ok<br>-1= manejador incorrecto | Avanza o retrocede el robot el número de metros "deltaçon velocidad "v". Cualquier llamada a VWDriveStraight, -Turn, -Curve o VWSetSpeed interrumpirá la ejecución de este comando. |
| int VWDriveTurn (VW-Handle handle, radians delta, radPerSec w) | (handle) Manejador VW-Driver<br>(delta) ángulo de giro en radianes (negativo dirección de las agujas del reloj).<br>(w) velocidad de giro (siempre positiva) | 0=ok<br>-1= manejador incorrecto | Hace girar el robot un ángulo "delta"en radianes con una velocidad "w". Cualquier llamada a VWDriveStraight, -Turn, -Curve o VWSetSpeed interrumpirá la ejecución de este comando. |
| int VWDriveCurve (VW-Handle handle, meter delta_l, radians delta_phi, meterpersec v) | (handle) Manejador VW-Driver<br>(delta_l) longitud del segmento de la curva en metros<br>(delta_phi) ángulo de giro en radianes (negativo dirección de las agujas del reloj)<br>(v) velocidad (siempre positiva) | 0=ok<br>-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 o VWSetSpeed interrumpirá la ejecución de este comando. |
| float VWDriveRemain (VWHandle handle) | (handle) Manejador VW-Driver | 0.0 = si el comando VW-Drive previo ha sido completado.<br>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 (VW-Handle handle) | (handle) Manejador VW-Driver. | -1= Manejador incorrecto<br>0 = El vehículo está todavía en movimiento.<br>1 = El comando previo VWDrivex ha sido completado. | Chequea si el comando previo VWDrivex ha sido completado |
| int VWStalled (VWHandle handle) | (handle) Manejador VW-Driver. | -1= Manejador incorrecto.<br>0 = El robot está todavía en movimiento o el comando que bloquea el movimiento está activo.<br>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 de la interfaz VW (II)

### 3.2.2. Acceso a la cámara.

El sistema operativo Robios ofrece una serie de funciones para tomar imágenes con la cámara, accediendo a esta desde el programa, según muestra el cuadro 3.3. Para acceder a la cámara, como en el caso de los motores de continua, es necesaria una inicialización que se realiza con la función "CAMInit(..)" a la que se pasa como único

parámetro el factor de zoom con el que se quieren obtener las imágenes. Después de esta inicialización ya pueden obtenerse las imágenes capturadas por la cámara, pero también hay que tener en cuenta que existe un periodo de estabilización de la cámara durante el cual las imágenes capturadas no son nítidas. La captura de imágenes puede hacerse de dos maneras según el tipo de imágenes que se desee obtener. Para obtener imágenes en escala de grises se utiliza la función "CAMGetFrame(..)" que tiene como único parámetro una imagen de 62 x 82 pixels. Para la obtención de imágenes en color se usa la función "CAMGetColorFrame(..)" a la que se le pasa una imagen de 62 x 82 x 3 bytes, cada pixel tiene tres valores ahora (los de las coordenadas RGB : Rojo, Verde, y Azul), y otro parámetro que indicara si se captura la imagen en color o la pasa automáticamente a escala de grises. Las imágenes en color no pueden visualizarse directamente en el EyeBot ya que el display del que dispone es en blanco y negro.

Para poder visualizar imágenes en color capturadas por la cámara del EyeBot se utilizara el programa Teleoperador [GarcíaMorata02]. Este programa permite la descarga al PC a través del puerto serie así como su visualización de imágenes,en color o blanco y negro, capturadas por el EyeBot.

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

Cuadro 3.3: Funciones para la cámara

También es posible decir si se quiere que se haga un ajuste automático del brillo con la función "CAMMode(..)" a la que se le pasa la constante AUTOBRIGHTNESS si se quiere ajuste automático, ó NOAUTOBRIGHTNESS en caso contrario.

Al finalizar la ejecución del programa deberá liberarse el uso de este recurso usando la función "CAMRelease( )", que no necesita parámetros.

### 3.2.3. Acceso a los servos.

Como ocurre con otros elementos hardware es necesaria una inicialización de estos con el comando "SERVOInit (nombre)" cuyo único parámetro es el nombre del servo que queremos utilizar. Para el comportamiento sigue pared solo es necesario el uso del servo que controla la posición de la cámara, para que esta este siempre orientada la derecha. Para fijar los servos en el ángulo deseado se utiliza la función "SERVOSet(servo,ángulo)", donde 'servo' se el nombre del manejador del servo que queremos utilizar, y 'ángulo' es el valor del ángulo de fijación del servo. En el cuadro 3.4 se detallan las funciones de acceso a los servos.

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

Cuadro 3.4: Funciones para los servos

Al final de la ejecución del programa también es necesario liberar su uso, con la llamada a la función "SERVORelease(servo)" cuyo parámetro es el nombre del manejador del servo utilizado.

### 3.2.4. Acceso al teclado.

Permiten la interacción del usuario con el robot (cuadro 3.5). De este modo las funciones que ofrece nos van a poder decir que tecla se pulsó a través de las función "KEYGet( )". También podremos saber si se pulsó una determinada tecla con la función "KEYRead( )" que lee la tecla para ver si se pulsó. Incluso parar la ejecución del programa hasta que se haya pulsado una determinada tecla con la función "KEYWait(tecla)", a la que se le pasa como parámetro el nombre de la tecla que debe pulsarse.

| *Formato Función* | *P.Entrada* | *P.Salida* | *Descripción* |
|---|---|---|---|
| Teclado | | | |
| KEYGetBuf (letra) | Puntero a un carácter. | Carácter con la tecla pulsada (KEY1, KEY2, KEY3, KEY4 de izq. a der.). | Espera que se presione una tecla y la almacena en letra. |
| KEYGet | Ninguno. | Código de la tecla pulsada (KEY1, KEY2, KEY3, KEY4 de izq. a der.) | Devuelve el valor de la tecla pulsada. |
| KEYRead | Ninguno. | Código de la tecla pulsada (KEY1, KEY2, KEY3, KEY4 de izq. a der.)<br>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 3.5: Funciones para el teclado

### 3.2.5. Acceso al LCD.

Nos permite representar diferentes tipos de datos en la pantalla según las funciones de el cuadro 3.6. Vamos a poder escribir cadenas de caracteres con la función "LCDPrintf(cadena)". Números tanto enteros como reales pueden ser escritos en pantalla con las funciones "LCDPutInt(entero)" y "LCDPutFloat(real)" respectivamente. De este modo podremos visualizar lo que ve el EyeBot en cada momento cuando esta desarrollando el comportamiento sigue pared.

La función "LCDPutImage(imagen)" que representa una imagen binaria completa de 124 x 68 pixels en blanco y negro, presentando la imagen sobre el LCD a pantalla completa. Mientras que la función "LCDPutGraphic(imagen)" escribe en la pantalla LCD una imagen en blanco y negro también pero de 82 x 62 pixels, corta la imagen en las filas inferiores.

| *Formato Función* | *P.Entrada* | *P.Salida* | *Descripción* |
|---|---|---|---|
| Pantalla | | | |
| LCDPrintf (texto) | Formato, cadena y parámetros. | Ninguno. | Imprime el texto en el LCD (versión simplificada del printf). |
| LCDClear | Ninguno. | Ninguno. | Limpia el LCD. |
| LCDPutChar (char) | Carácter a escribir | Ninguno. | Imprime el carácter en la posición del cursor. |
| LCDSetChar (fila, columna, char) | Carácter a escribir.<br>Número de la fila (0-15).<br>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.<br>Número de la fila (0-15).<br>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.<br>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.<br>Mínimo número de espacios.<br>Número de decimales después del punto | Ninguno. | Escribe el real con espacios delante y con los decimales indicados. |
| LCDMode (modo) | Modo de display deseado.<br>(NON)SCROLLING.<br>(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).<br>Columna (0-15). | Ninguno. | Coloca el cursor en la posición dada. |
| LCDGetPos (fila, columna) | Punteros a enteros que almacenarán la fila y la columna. | Fila actual (0-6).<br>Columna actual (0-15). | Devuelve la posición en la que está el cursor. |
| LCDPutGraphic (imagen) | Imagen en blanco y negro. | Ninguno. | Escribe la imagen en blanco y negro en el LCD empezando por la esquina superior izquierda. Sólo son escritos 80x54 píxels. |
| LCDPutColorGraphic (colorimag) | Imagen en color. | Ninguno. | Escribe la imagen en color empezando por la esquina superior izquierda. Sólo son escritos 80*54 píxels. CUIDADO: utilizando esta función se destruye el contenido de la imagen. |

Cuadro 3.6: Funciones para la pantalla

Otra de las funciones para visualizar imágenes en la pantalla LCD es la función "LCDPutColorGraphic(imagen)" que representa una imagen en color en varias tonalidades de grises que consigue alternando en el tiempo blancos y negros.

### 3.2.6. Recursos de Multiprogramación.

El sistema operativo del EyeBot permite la multiprogramación de dos maneras distintas, con desalojo y sin desalojo (método cooperativo). En el modo cooperativo sólo se ejecuta aquella tarea que tiene el testigo, de manera que si se bloquea ninguna otra tarea podrá ejecutarse, porque el control de flujo está controlado por la tarea bloqueada.

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

Para trabajar en modo multitarea con el EyeBot es necesario elegir el método de multiprogramación que se necesite, siendo el que más se ajusta a nuestras necesidades el método con desalojo por lo anteriormente comentado. Las funciones para trabajar con tareas están en el cuadro 3.7, donde el parámetro 'thread'es un puntero a la estructura 'tbl' o 0 para el thread actual.

Esta posibilidad de trabajo en multiprogramación será muy útil para el desarrollo del comportamiento sigue pared mediante visión ya que podremos tener hebras o flujos de ejecución distintos que se encarguen de tareas especificas, lo que hará que este comportamiento sea mas reactivo y dinámico.

| *Formato Función* | *P.Entrada* | *P.Salida* | *Descripción* |
|---|---|---|---|
| Tareas | | | |
| OSMTInit (modo) | Modo de operación. COOP (por defecto). PREEMT | Ninguno. | Inicializa el entorno multitarea |
| OSSpawn (nombre, dircom, tampil, prioridad, uid) | Nombre de la tarea Dirección de comienzo. Tamaño de su pila. Prioridad(MINPRI-MAXPRI). Identificador | Puntero al bloque de control de la tarea inicializada. | Devuelve el thread inicializado e insertado en el planificador pero no puesto a listo. |
| OSMTStatus | Ninguno. | Modo multitarea PREEMPT, COOP, NOTASK | Devuelve el modo de la actual multitarea. |
| OSReady (thread) | Puntero al bloque de control de una tarea. | Ninguno. | Pone el thread a listo. |
| OSSuspend (thread) | Puntero al bloque de control de una tarea. | Ninguno. | Pone el valor del thread a suspendido. |
| OSReschedule | Ninguno | Ninguno | Elige una nueva tarea. |
| OSYield | Ninguno. | Ninguno. | Suspende el actual thread y replanifica. |
| OSRun (thread) | Puntero al bloque de control de una tarea. | Ninguno. | Pone a listo el thread dado y replanifica. |
| OSGetUID (thread) | Puntero al bloque de control de una tarea. | UID de la tarea. | Devuelve el identificador del thread dado. |
| OSKill (thread) | Puntero al bloque de control de una tarea. | Ninguno. | Elimina el thread pasado y replanifica. |
| OSExit (código) | Código de salida. | Ninguno. | Mata el thread actual con el código de salida y mensaje. |
| OSPanic (mensaje) | Mensaje de texto | Ninguno. | Cuando se produce un error, imprime el mensaje y para el procesador. |
| OSSleep (tiempo) | Tiempo en centésimas de segundo (1/100). | Ninguno. | Permite al thread pararse durante el tiempo indicado, en multitarea se pasa el control a otro thread, es una llamada a OSWait en monotarea. |
| OSForbid | Ninguno. | Ninguno. | Desactiva el thread pasando a modo PREEMPT. |
| OSPermit | Ninguno. | Ninguno. | Activa el thread pasando a modo PREEMPT. |
| | | | Thread es un puntero a la estructura tbl o 0 para el thread actual |

Cuadro 3.7: Funciones para las tareas en la multiprogramación.

Otro recurso de multiprogramación que vendrá bien para la generación del comportamiento son los semáforos (cuadro 3.8) que se utilizaran para coordinar las distintas tareas y garantizar el acceso en exclusión mutua a variables compartidas. El acceso a ellos para modificar su estado requiere su inicialización a través de la función "OSSemInit(sem,valor)". Una vez inicializados pueden ser subidos o bajados con las funciones "OSSemP(sem)" y "OSSemV(sem)" respectivamente. Si el semáforo está subido la tarea que controle no se podrá ejecutar hasta que sea bajado.

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

Cuadro 3.8: Funciones para los semáforos

## 3.3. COMPILACIÓN CRUZADA.

Los programas que se ejecutan en el EyeBot se desarrollan en un entorno de programación determinado, que desarrolla a través del PC y del EyeBot.

Una vez creado el entorno de programación ya podremos editar programas fuente para el robot, estos programas son creados en el PC. Se pueden escribir en cualquier editor de texto, y se codifican en lenguaje ANSI-C, para posteriormente compilarse este código fuente con el compilador cruzado *gcc68* , que a partir de código fuente en lenguaje ANSI-C genera un archivo ejecutable (*ejecutable.hex*) para el microprocesador Motorola del EyeBot. Una vez que se tiene este código ejecutable se desde el PC al EyeBot a través del puerto serie. Tras esta descarga se tiene la posibilidad de almacenar el ejecutable en la memoria ROM del EyeBot, si no se guarda al apagar el robot el código cargado en la memoria RAM se borrara (figura 3.4).

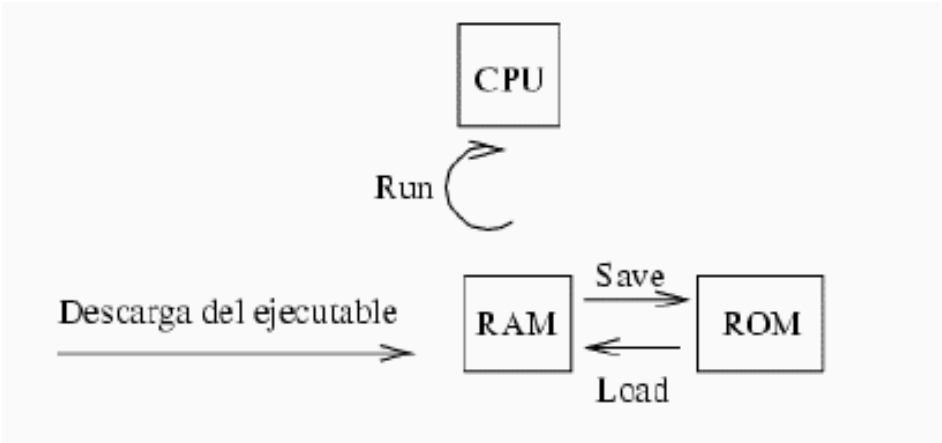


Figura 3.4: Esquema de descarga del código ejecutable

Este entorno de programación se consigue instalando el compilador cruzado de genera código ejecutable para el EyeBot, este compilador se llama gcc68 y se puede

conseguir en la pagina oficial del EyeBot [1]. Para obtener mas detalles sobre estas herramientas se puede consultar el manual del EyeBot [Cañas02a]. En la figura 3.4 se puede observar esquematizadamente los pasos a seguir para llevar a cabo la descarga del programa ejecutable.

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

# Capítulo 4

# DESCRIPCIÓN INFORMÁTICA.

En este capítulo se analizará la implementación informática del comportamiento sigue pared desarrollado para el robot móvil EyeBot. Se hará una descripción de la implementación a nivel estructural, es decir cómo se ha divido el comportamiento en tareas más sencillas y específicas en función de la arquitectura escogida, la arquitectura de control JDE [Cañas02b].
Para describir la implementación se comenzara dando visión general de esta, donde se especifica su división en diferentes hebras. Para a continuación pasar a describir más en detalle cada una de ellas.

## 4.1. ESTRUCTURA GENERAL.

La estructura general de la implementación esta basada en la arquitectura JDE (*Jerarquía Dinámica de Esquemas*) mencionada en el capítulo 1. Se trata de una arquitectura de control para robots autónomos, que basa el desarrollo de comportamientos para estos robots, en la división de este comportamiento en diferentes esquemas. Estos esquemas se encargaran cada uno de la realización de una tarea especifica.

Dentro de la arquitectura JDE existen dos tipos de esquemas , esquemas perceptivos que como su nombre indica se encargaran de recoger los estímulos del entorno para su proceso y posterior interpretación. Y por otra parte están los esquemas de control, cuya misión es generar ordenes de control para el robot en función de la información generada por los esquemas perceptivos. De este modo para llevar a cabo este planteamiento es necesaria la comunicación entre estos esquemas que generan el comportamiento para el robot autónomo.

Atendiendo a las características de esta arquitectura de control se llevó a cabo el diseño del comportamiento en dos esquemas separados, uno perceptivo y otro de ac-

tuación según ilustra la figura 4.1. El esquema perceptivo será el encargado de capturar imágenes a través de la cámara del EyeBot para posteriormente procesarlas y extraer de estas la información relevante y necesaria. El esquema de control del robot decide la orden que debe comunicar al EyeBot para que actué de una manera u otra en base a esa información.

La información que hemos encontrado más relevante para el comportamiento sigue pared es la línea frontera entre el suelo y la pared. Se percibirá utilizando distintas técnicas de tratamiento de imágenes, como filtros, segmentación, etc.

Por su parte el esquema de control del comportamiento se encargara de dictar las ordenes de control pertinentes al robot EyeBot para que actúe de la manera más adecuada, en función de la información extraída por el esquema perceptivo. De modo que deberá analizar la línea frontera para determinar ante que caso se encuentra. Por ejemplo si esta alineado correctamente con la pared que sigue, si se ha desviado a la hacia un lado y debe corregir su trayectoria, si solo ve pared en las imágenes para girar y evitar colisionar, etc.


Figura 4.1: Arquitectura del comportamiento.

Este diseño conceptual en esquemas se materializa desde el punto de vista informático en un programa. De modo que estos esquemas conceptuales se implementan como distintos flujos o hilos de ejecución. Que se encargan de llevar a cabo las tareas específicas que realiza cada esquema.

Afortunadamente el sistema operativo del EyeBot ofrece la posibilidad de llevar a cabo la implementación de este diseño. Gracias a las facilidades que ofrece para trabajar con multiprogramación como se vio en el capítulo 3.

De este modo la implementación del comportamiento usa tres hebras o hilos de ejecución como muestra la figura 4.2. Una hebra que materializa el esquema perceptivo, otra hebra encargada del control del robot y por último una hebra de apoyo que ayuda

a depurar el comportamiento. Esta última se encarga de chequear la botonera, situada debajo de la pantalla LCD. Para ver si se pulso la tecla etiquetada como *FIN*, lo que da por finalizada la ejecución del programa que implementa el comportamiento, o para ver si se pulso otra tecla para elegir entre las distintas opciones de visualización en la pantalla del robot.

La comunicación entre las distintas hebras se realiza a través de variables globales, a las que todas las tareas tendrán acceso. Para evitar posibles incoherencias, en los valores de estas variables globales, se utilizan semáforos para asegurar que en cada momento solo pueda acceder a estas variables una sola tarea. La utilización de los semáforos, para la implementación del comportamiento, es otro de los recursos de multiprogramación que nos ofrece la interfaz de programación del robot.

Además de estas hebras que se encargan de llevar a cabo el comportamiento a través del programa, antes de activar estas hebras en la implementación existe también una fase de calibración que se incluye dentro del bloque de inicialización, y que se realiza antes de que el robot vaya a seguir la pared. En este fase de calibración, entre otras cosas, se gira la cámara del EyeBot hacia la derecha con un determinado ángulo de visión sobre la pared. Esto se realiza accediendo al servo que controla la cámara a través de la interfaz de programación. También se determina, en esta fase de calibración, a que distancia de la pared lateral que seguirá queremos que el EyeBot desarrolle el comportamiento de seguir la pared. De tal modo que establezcamos ya una referencia, para la posterior navegación del robot por el entorno.

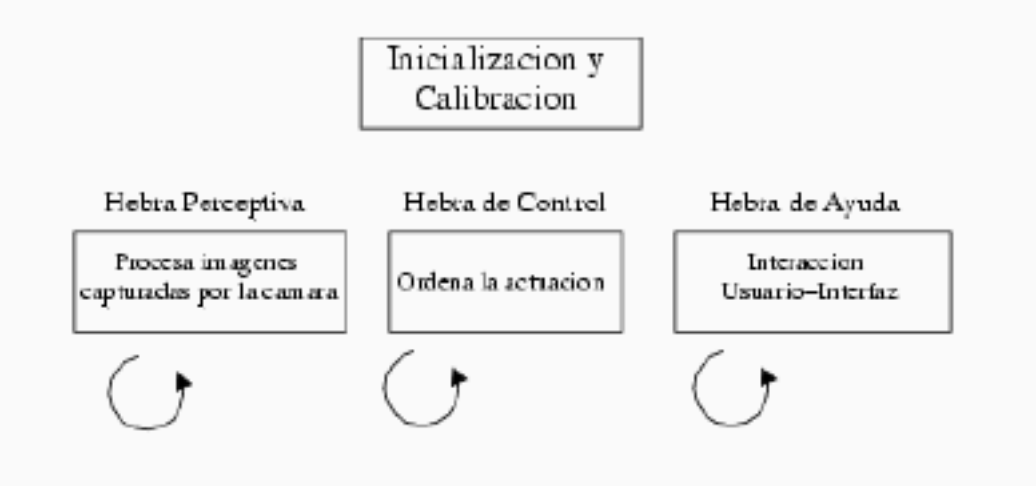


Figura 4.2: Estructura de la implementación.

En los siguientes apartados se van a detallar estas tareas o hebras que implementan los esquemas conceptuales que componen el comportamiento sigue pared mediante visión, basado en la arquitectura JDE.

## 4.2. ESQUEMA PERCEPTIVO.

En este apartado se va a hacer una descripción del esquema perceptivo que, como se comento en el apartado anterior, se implementa como un hilo de ejecución o hebra. Básicamente lo que tiene que hacer es percibir estímulos relevantes del entorno a través de la cámara del EyeBot. La información más relevante es la línea frontera entre el suelo y el resto de objetos. Para que el robot móvil sea capaz de desenvolverse a través del entorno, en función de como vea esta línea, si aparece o no, si tiene una determinada orientación,etc.

El procesado de estas imágenes capturadas por el robot a través de la cámara se divide en tres fases o etapas claramente diferenciadas. Una primera fase en la que se discrimina en las imágenes el suelo del resto de los objetos, que se lleva a cabo realizando un filtrado por color de la imágenes, devolviéndonos las imágenes binarizadas. La siguiente fase define los puntos que componen la línea frontera a partir de la imagen filtrada. Y por último en la tercera fase de procesado se hace una segmentación de la línea frontera de la imagen obtenida en la fase anterior, de modo que caractericemos esta por los segmentos que la componen.

Estas fases de tratamiento de las imágenes se implementan en el programa como funciones que nos devuelven los datos relevantes requeridos en cada fase. A continuación se realiza una descripción en detalle de la implementación de estas.

### 4.2.1. Filtro de color.

Al capturar una imagen lo primero que necesitamos es identificar claramente en esta, el suelo por un lado y por otro los elementos verticales (paredes,puertas,etc.) para poder diferenciar estos elementos uno del otro, el suelo de los elementos verticales, debemos buscar invariantes perceptivas. La invariante perceptiva que nos va hacer diferenciar el suelo del resto de elementos del mundo exterior es el color del propio suelo.

Esta fase se implementa como una función que recibe como entrada una imagen en coordenadas RGB y devuelve una imagen binarizada que clasifica cada pixel como suelo o no.

Este filtro conviene que sea lo más robusto y eficiente posible, y que en condiciones variables de luminosidad, etc. sea capaz de diferenciar bien el suelo del resto de objetos.

La cámara con la que esta dotada el EyeBot captura las imágenes en color en espacio de coordenadas RGB, este espacio trabaja almacenado cada pixel de los que se compone la imagen como la combinación tres colores, el rojo ('R'), el verde ('G') y el azul ('B'). En teoría las características particulares de este espacio de coordenadas lo hacen poco indicado para trabajar con él a la hora de realizar un filtro de color. Esto es debido a que es altamente sensible a condiciones variables de luminosidad, los valores RGB para un mismo color son muy diferentes si cambia la intensidad luminosa. Este hecho se pudo comprobar tomando imágenes en color a través de la cámara del EyeBot y descargándolas en el PC a través del puerto serie. Visualizando el fichero correspondiente a las imágenes en color, capturadas por el EyeBot se podían observar los valores de las coordenadas RGB para esas imágenes del entorno observándose que para imágenes localizadas en el mismo lugar pero con distintas condiciones de luz, el valor de las coordenadas en RGB para el suelo era excesivamente variable.

Esta tarea se llevo a cabo utilizando el proyecto Teleoperador [GarcíaMorata02] que entre otras funcionales ofrece la posibilidad de grabar imágenes, en color o blanco y negro, capturadas por el EyeBot. Sirviéndonos de este Teleoperador para depurar la implementación de estas fases de tratamiento de imágenes.

Para intentar solventar este inconveniente de trabajar con las imágenes en color sobre un espacio de coordenadas RGB, se pensó que seria mas interesante trabajar con las imágenes sobre otro espacio de color que no se viera tan influido por el cambio en las condiciones de luz. Un espacio más invariante a cambios de intensidad luminosa es el espacio HSI, el cual basa las imágenes en color como una combinación de tres valores, el brillo ($H$ o Hue), la saturación ($S$ o Sat) y la intensidad ($I$ o Int). Un problema de este espacio de color es que la cámara entrega coordenadas RGB, por lo tanto habría que hacer una transformación de las imágenes en color en RGB al espacio HSI. Estas conversión obedece a las siguientes ecuaciones.

$$H = cos^{-1}\left(\frac{\frac{(R-G)+(R-B)}{2}}{\sqrt{(R-G)^2+(R-B)(G-B)}}\right)$$

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

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

Se estudio la posibilidad de implementar el filtro basado en coordenadas HSI, que era algo más robusto frente a condiciones variables de luminosidad los resultados obtenidos no eran los aceptados por el tiempo de computo que requiere. Incumpliendo el requisito de vivacidad a la hora de llevar a cabo el comportamiento. Esto es debido a que para el calculo de la coordenada $H$ del espacio HSI es necesario aplicar una función trigonométrica, lo que hace al procesador realizar un calculo mayor y gastar mas tiempo de proceso, por cada pixel de los que componen la imagen. Esto supone ralentizar el tiempo de procesado de las imágenes y con este la vivacidad deseada en los movimientos del robot, que estaría más tiempo deliberando y procesando que actuando. Cuando lo que se busca de este comportamiento es que sea dinámico y reactivo.

De este modo la alternativa elegida para llevar acabo este filtro de color fue el trabajar directamente con las imágenes en coordenadas RGB, pero manteniendo controladas las condiciones de luminosidad sobre las cuales el EyeBot llevaría a cabo el desarrollo del comportamiento sigue pared mediante visión.

Para llevar a cabo el filtrado de las imágenes en función del color del suelo, es necesario antes conocer los valores de las coordenadas RGB cuando equivalen al color del suelo, teniendo en cuenta además que estos valores deben ser correspondientes a dicho color del suelo bajo las condiciones de luz controlada, en las cuales se desarrolla el comportamiento. Estos valores se ajustaron manualmente grabando imágenes del entorno.

El filtro de color realiza una pasada por todos los pixels que componen una imagen en color, comparando en cada uno de estos si sus coordenadas en RGB se encuentran dentro del rango de valores que se considera como color del suelo. Además de esta comparación se realizan otras dos comparaciones para clasificar un pixel como suelo o no. Las siguientes ecuaciones ilustran este proceso.

$$
\begin{aligned}
60 >= R <= 140 \\
70 >= G <= 150 \\
50 >= B <= 100 \\
R > G \\
B > R
\end{aligned}
$$

Implementando el filtro a través de simples comparaciones de enteros, que es como se almacena el valor de cada coordenada RGB, evitamos el trabajo con tipos de datos reales (*float*). Con lo que se consigue es que el procesado de la imagen sea más rápido. Ya que si se realiza alguna operación con un número real por cada uno de los pixel que compone la imagen (62x82 pixels) para filtrar esta, se produce un retardo que se ve multiplicado por el número de pixels que componen la imagen. Debido a que el microprocesador del robot no tiene coprocesador matemático, por lo que emula todas las operaciones con números reales. De esta manera el código que implementa este filtro de color también es más sencillo.

Las imágenes 4.3 ilustran el efecto de aplicar el filtro de color sobre una imagen del entorno donde se desenvuelve el robot. Estas imágenes fueron capturadas por el EyeBot, pero descargadas en el PC.

Figura 4.3: Imagen sin filtrar y filtrada del entorno

### 4.2.2. Establecer la frontera entre el suelo y las paredes.

Una vez que se ha conseguido obtener unas imágenes en las que se diferencian claramente el suelo del resto de objetos, el siguiente paso es tener claramente definida la línea frontera o línea intersección entre el suelo y la pared. De este modo se puede calcular la posición y orientación del robot con respecto a esta línea.

Para llevar esto a cabo se hace un recorrido por cada una de las columnas de pixels de las que se compone cada imagen ya filtrada, buscando el punto donde acaba el suelo o comienza la pared. Este recorrido de las columnas que componen las imágenes se lleva a cabo de abajo hacia arriba, ya que si en las imágenes capturadas se visualiza el suelo este aparece en la parte inferior de estas. Así lo que nos interesa es buscar donde acaba este suelo en las imágenes capturadas, delimitar la frontera. De este modo en cada columna vamos mirando de abajo a arriba cada pixel para ver si es suelo, si

no lo es debe de ser el primer pixel perteneciente a la pared de modo que guardamos la posición que ocupaba, ya que será uno de los puntos que componen la frontera, y abandonamos el recorrido a través de esa columna. Esta operación se repite para todas las columnas.

Es necesario tener en cuenta el ruido que puede haber en las imágenes filtradas, es decir pixels aislados que no sean del suelo en la parte de la imagen considerada como suelo. Teniendo en cuenta esta consideración la implementación del algoritmo que calcula la frontera debe soportar este ruido. Para solucionar este problema, y conseguir una solución más robusta y resistente al ruido, se introdujo al algoritmo expuesto anteriormente una condición adicional para aceptar un pixel, que no fuera del suelo, como punto frontera dentro de cada columna de las que esta compuesta cada imagen. Para verificar la hipótesis de que es el punto frontera de esa columna es necesario que los tres pixels siguientes a él en esa columna no sean suelo también. En caso contrario se sigue analizando la columna descartando ese pixel ya que será un pixel aislado de los que genera el ruido en la imagen.

Esta condición para evitar el ruido en las imágenes, se implementa a través de estados. Al comienzo del recorrido de cada columna de las imágenes partimos del denominado estado cero, de modo que pasamos al estado uno cuando encontramos el primer pixel que no sea suelo, y además guardamos su posición dentro de la columna donde se encontraba, ya que puede que sea el punto frontera de esa columna. Si tras este pixel los siguientes no son del suelo también, así hasta contabilizar tres más, llegamos al estado cuatro en el que verificamos la hipótesis de que el primer pixel, que nos hizo pasar del estado cero al estado uno, es el pixel o punto frontera de esa columna. En caso contrario de que no lleguemos al estado cuatro, será porque alguno de los tres pixels siguientes al primero de no suelo, eran del suelo y entonces el primer pixel de no suelo era aislado. De modo que se vuelve al estado cero y se sigue analizando la columna.

A continuación se ofrece el fragmento de código donde se implementa en la función la parte encargada de buscar el punto frontera saltándose pixels considerados como ruido. En el se puede ver como se pasa por distintos estados, en concreto desde el cero hasta el cuatro lo que hará que pueda solventar el ruido en las imágenes de hasta tres pixels aislados.

```
frontera=limitef;
estado=0;
```

```
  for(i=limitef; i>=0; i--) /* recorrido dentro de una columna */
     {
     if (encontrada == 0) /* no ha encontrado la frontera */
{
          if (estado==0)
     {
if (img[i][j]==suelono)
  {estado=1;
  frontera=i;}
   else
  frontera=i;
      }
   else if (estado==1)
         {if (img[i][j]==suelono) estado=2;
        else /*pixel aislado */
 {estado=0;img[frontera][j]=suelo;frontera=i;}
.
.
.
else if (estado==4)
      break;}
}
      }/* final del bucle que se recorre una columna de abajo a arriba */
    encontrada = 0;
    /* almacenamos el punto frontera en el vector de puntos frontera*/
    vf[j] = frontera;
.
.
```

Una vez determinada esta línea frontera, entre el suelo y los objetos verticales, ya disponemos de cierta información a cerca de la posición del EyeBot respecto de las paredes (figura 4.4). Podemos obtener de esta línea frontera el punto de mayor altura de esta frontera en la imagen, y el de menor altura. De tal modo que si la frontera esta compuesta por un solo segmento podemos determinar la orientación del robot respecto a la pared. Y con esta determinar algunas acciones de control para mover el robot.

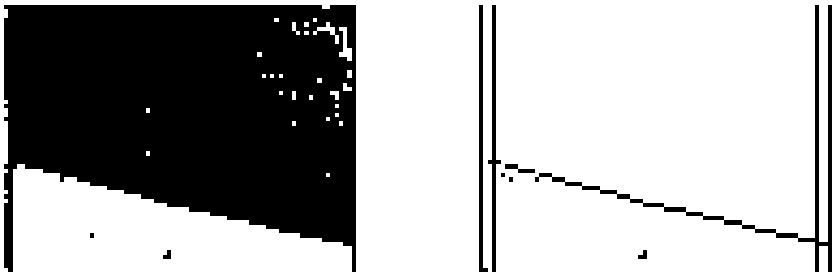

Figura 4.4: Imagen filtrada e imagen con línea frontera

Pero con esta información no es suficiente ya que si la imagen es la de una esquina cóncava o convexa, solo con los valores del punto más alto de la frontera y el más bajo, el robot no será capaz de reconocer que esta visualizando ese tipo de imágenes y las ordenes de control para que se mueva no serán las adecuadas.

Por este motivo una vez establecida la línea frontera de cada imagen, es necesario someter esta imagen obtenida a otra fase de procesamiento en la cual determinamos los segmentos que componen esta frontera.

### 4.2.3. Segmentación de la línea frontera.

Una vez obtenida la imagen filtrada por el color del suelo y determinados la línea de puntos que establecen la frontera entre el suelo y las paredes. Lo que necesitamos, para determinar con mayor precisión la orientación relativa del EyeBot con respecto a las paredes, son los segmentos, con distinta orientación, que componen esta frontera. En función de estos segmentos se puede determinar la situación del EyeBot de cara al comportamiento sigue pared. Por ejemplo si solo observa un segmento totalmente horizontal centrado en la imagen y por lo tanto esta de frente a la pared, si ve dos segmentos con orientaciones opuestas y por lo tanto se encuentra frente a una esquina, etc.

Para la implementación de una función que realiza esta segmentación hemos tenido en cuenta los requisitos descritos en el capítulo 2. La solución deberá ser lo más robusta posible pero rápida a la vez, y que le lleve poco tiempo de proceso al robot para no ralentizar las acciones de control. Teniendo en cuenta estas consideraciones se estudiaron distintos algoritmos o soluciones para la implementación de la función que llevaría a cabo la segmentación de la línea frontera.

Como primera aproximación se analizó la primera derivada de la línea frontera en

cada punto, es decir analizar el incremento del valor de la coordenada del eje $Y$ de cada punto de los que componen la línea frontera. De este modo cuando los valores de este incremento cambiaran significativamente de un punto a otro en algún punto se podría predecir por tanto acababa un segmento y comenzaba otro.

El problema de esta solución es que las rectas fronteras obtenidas no son ideales como se observa en la figura 4.5. Es decir están compuestas por escalones horizontales, que son puntos de la línea frontera que tienen el mismo valor en el eje de coordenadas $Y$. Al visualizar estos escalones juntos dan la impresión de formar una recta con una determinada orientación.

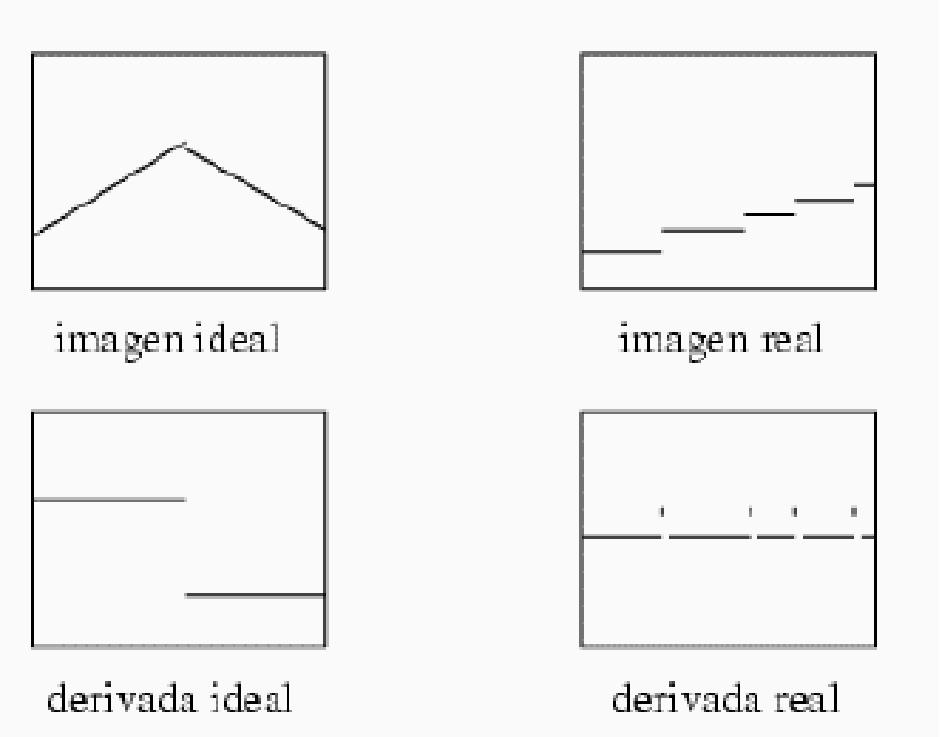


Figura 4.5: Primera derivada ideal y real de la recta.

La siguiente estrategia estudiada, se basaba en ir haciendo una estimación de la derivada en cada punto, a la derecha y a la izquierda de este. Esta estimación estaba basada en que un número de pixels, tanto a la derecha como a la izquierda de un punto, el valor de los incrementos y el de la frecuencia con el que se producían estos debían ser parecidos. En caso contrario sería indicativo de que el punto sobre el que se realizaba el análisis, a derecha e izquierda, era un punto final de un segmento y comienzo de otro si aun quedaban más puntos tras él.

Esta solución era capaz de segmentar la línea frontera pero no era la mejor para sobreponerse de ruido en las imágenes y por tanto no era la solución más robusta, ya que era demasiado sensible al ruido. En general a este método y al anterior, ambos basados en la derivada de la frontera, le afectaban mucho los puntos frontera aislados, que no se habían determinado con total precisión debido al ruido que había en las imágenes sobre las que se había realizado el filtro de color.

Una vez descartados estos dos algoritmos, para la posible implementación de la función que lleva a cabo la segmentación de la línea frontera. Se busco otra estrategia de trabajo basada en el crecimiento de segmentos. Esta solución consistía en calcular la pendiente de los dos primeros escalones que componían la línea frontera, de tal modo que se asumía que esta pendiente era la correcta para el primer segmento de la línea. Por lo tanto se hacia el calculo del valor hipotético en el eje de coordenadas $Y$ que debería tener cada punto de la línea asumiendo que la pendiente calculada inicialmente era la buena. De tal modo que el valor real en $Y$ de los puntos debía estar dentro de una franja de valores creada en función de la pendiente calculada inicialmente.

De este modo cuando al analizar un punto, su valor real en $Y$ no se parecía al que el correspondía con el valor hipotético calculado, se salia de la denominada franja de valores. Si no era considerado como punto aislado o ruidoso, se asumía que era punto final de un segmento e inicial del siguiente. A continuación se seguía aplicando el algoritmo para el resto de puntos que quedaban por analizar. Calculándose de nuevo una pendiente inicial a partir del punto donde termino el segmento anterior.

De esta manera se iban extrayendo los segmentos de izquierda a derecha de la imagen, tal y como se analizaba la imagen, con este algoritmo lo que se hacia era construcción de hipótesis. El valor calculado inicialmente de la pendiente, se asumía como un valor correcto. De este modo todos los cálculos posteriores estaban en función de este valor, así que este debía ser lo más preciso posible. Pero se pudo comprobar que esta solución era demasiado sensible por este motivo justamente, dependía demasiado de los valores iniciales y estos no eran seguros. No siempre el valor de la pendiente era el correcto y cuando esto sucedía la segmentación era imprecisa, se obtenían demasiados segmentos.

De este modo se llego al estudio de otro algoritmo basado en el principio contrario al anterior, no construía segmentos sino que hipotetizaba sobre ellos, no era incremental

sino decremental.

Este algoritmo consiste en estimar la pendiente entre el primer y el último pixel de la línea frontera, para posteriormente intentar verificar si todos los puntos están, con una determinada tolerancia, dentro esta recta hipotetizada. Esto se realiza como en la versión anterior. De este modo se calcula la posición hipotética sobre el eje de coordenadas $Y$ para cada punto $x$ del interior basándonos en la ecuación punto-pendiente de la recta mostrada a continuación. Por la cual podemos calcular el valor de $y$ ($y\_hipotetico$) para un punto si conocemos el valor $x$ para ese punto ($x$), otro punto de la recta ($x1$,$y1$) donde se encuentra el punto a analizar y el valor de la pendiente ($pendiente\_recta$) para la recta.

$$y\_hipotetico = pendiente\_recta * (x - x1) + y1$$

Una vez calculado este valor se compara con el valor real que tiene ese punto en el eje $Y$, si no son parecidos se sale de la franja y puede asumirse como punto inicial de un segmento de manera que se para hay el algoritmo para ahora trabajar con un nuevo segmento que será el que tenia como punto inicial al que su valor del eje $Y$ daba fuera de la franja establecida, y como punto final al de más a la derecha de la imagen. Es decir el segmento de más a la derecha, de este modo este algoritmo extrae los segmentos de la línea frontera de derecha a izquierda. Esta solución asume que un segmento es bueno, hace valida la hipótesis, cuando al pasarle un segmento todos los puntos que pertenecen a este al ser analizados están dentro de la franja, para ese segmento.

Cuando se valida una de estas hipótesis, se extrae un segmento, quiere decir que hemos encontrado algún segmento por la derecha de la imagen. Entonces el siguiente paso es pasar a analizar el resto de línea frontera que queda sin explicar, así ahora el algoritmo trabajara con un segmento de punto inicial el que esta más a la izquierda de la imagen, al comienzo de esta, y como punto final el punto que se encuentra a la izquierda del punto inicial del segmento que acabamos de obtener. Estas operaciones se repetirán hasta que se haya analizado toda la línea frontera como se muestra en la figura 4.6.

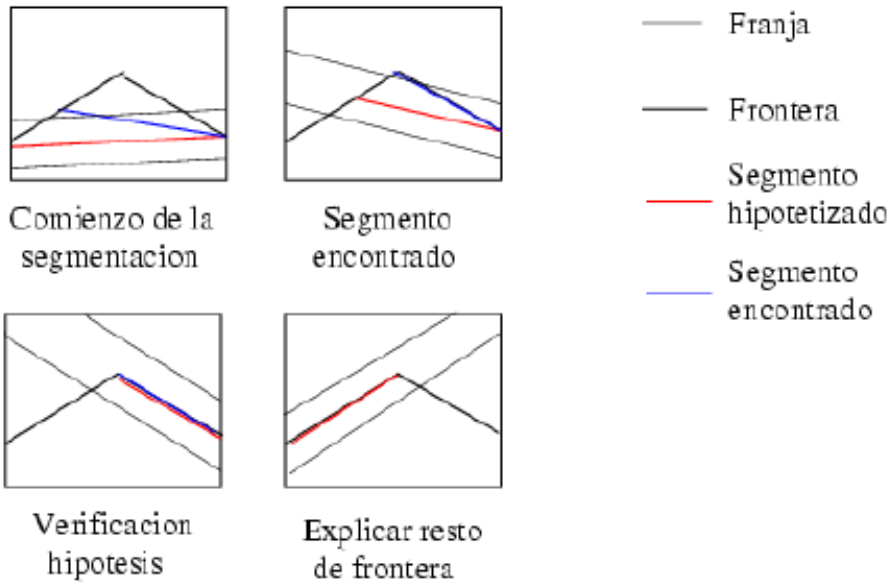


Figura 4.6: Proceso de segmentación.

El efecto de esta segmentación sobre las imágenes se puede observar en la figura 4.7.

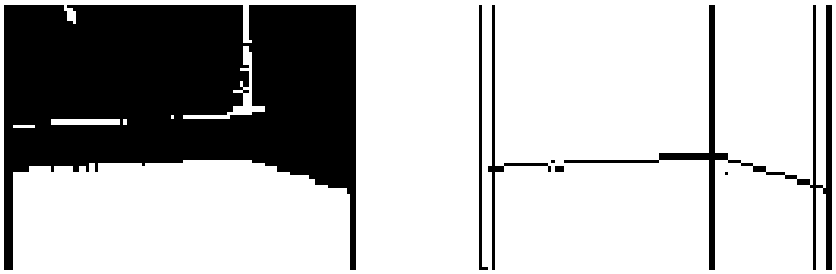

Figura 4.7: Imagen filtrada e imagen con línea frontera segmentada

Esta solución para la segmentación de la imagen es la más robusta y precisa de todas las evaluadas, ya que además es tolerante a tres pixels aislados. Pero aun así es algo sensible al ruido, de modo que tras obtener todos los segmentos que componen la línea frontera, se analizaran estos en busca de posible puntos de ruido. Esto sucede cuando existe un punto de la frontera que no se encontraba dentro de la franja, pero era un punto aislado, de modo que dividió lo que consideraremos un solo segmento en dos con idéntica inclinación o pendiente y que en realidad deben de formar un único segmento.

Para corregir estos errores se realiza una integración de estos segmentos que se pue-

den considerar como uno solo. Para llevar a cabo esta integración lo que se hace es comparar segmentos vecinos para ver si son cercanos tanto en el eje $X$ como en el $Y$ y además poseen una pendiente parecida. Si cumplen estas condiciones se podrán integrar, ya que lo más seguro es que entre medias hubiera un punto aislado que dividió lo que consideramos como un segmento solo, en dos segmentos.

Una vez terminada esta segmentación incluida la fase de integrar segmentos, la información de la que disponemos es lo bastante fiable y precisa como para empezar a emitir ordenes de control. De esta misión se encargara el esquema de control, que se implementa también mediante hebras y que será descrito a continuación.

## 4.3. ESQUEMA DE CONTROL.

La implementación del esquema de control se realiza, como en el caso de el esquema perceptivo, mediante una hebra. Esta hebra es la encargada de ordenar el movimiento al robot y basa sus decisiones en el análisis de distintos casos o situaciones que se le pueden presentar al EyeBot durante el desarrollo del comportamiento.

De este modo esta hebra que dicta las pertinentes ordenes de movimiento, indica una acción u otra en función del caso o situación que se este produciendo en ese momento. Para determinar ante que se situación se encuentra el robot la hebra de control estudia la información relevante extraída por la hebra perceptiva, la caracterización de la línea frontera entre el suelo y la pared.

Los movimientos que la hebra de control indica al EyeBot se implementan usando las funciones de acceso a los motores de la librería del control VW. Más concretamente las funciones que permiten el control en velocidad del robot, que hacen el movimiento del robot más suave que si usáramos el control en posición que hace que cuando el robot recorre una determinada distancia se pare. De modo que para mover el robot EyeBot utilizaremos la función de control en velocidad ''VWSetSpeed(..)" descrita en el cuadro 3.1 del capítulo 3.

Analizando el control desde este punto de vista, tenemos que el EyeBot tiene como objetivo seguir la pared lateral situada a su derecha. Por lo tanto el segmento más representativo, de los que componen la línea frontera de cada imagen, es el situado más a la derecha. De esta manera para el robot es de algún modo el *segmento guia* en función del cual actua de una manera u otra. Este control se realiza mediante el análisis de los siguientes casos:

- Los casos extremos en los que la línea frontera solo se divide en un segmento y además este está arriba del todo en la imagen o abajo del todo como se muestra en la figura 4.8. Para reconocer estos casos se comprueban los valores de las coordenadas $y$ de los dos puntos que caracterizan al segmento. En las imágenes el valor cero para la coordenada $y$ está en la parte superior de estas, mientras que la parte inferior se corresponde con el valor máximo. De tal modo que si las coordenadas $y$ del segmento son igual a cero, esto implica que el segmento estará arriba del todo y entonces el robot esta ante el caso en el que solo ve el suelo, por lo tanto deberá girar hacia la derecha para buscar la pared que sigue. Mientras que si por contra estas coordenadas tienen el valor máximo (*número de filas de la imagen*), esto significa que el robot solo visualiza pared, y en este caso deberá evitar chocar con esta girando hacia la izquierda rápidamente.

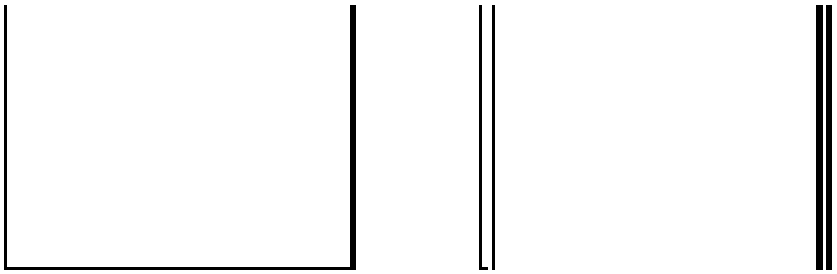

Figura 4.8: Visualización de los segmentos en los casos todo pared y todo suelo.

- El caso en el que la línea frontera este compuesta por un gran número de segmentos, y además quede una gran parte de esta línea frontera sin explicar. Es decir que el valor de la suma de las longitudes de los segmentos que componen la frontera no son ni la mitad del valor máximo (*número de columnas de la imagen*) que puede tener la línea frontera. Este caso se produce cuando en las imágenes capturadas por el robot hay demasiado ruido, generalmente porque esta enfocando a puntos lejanos espacialmente que captura con poca precisión. Este caso se denomina de *discontinuidad espacial*.

  Esta situación de *discontinuidad espacial* se le presentara al robot generalmente cuando se aleje demasiado de la pared lateral que sigue, se separe de esta como se observa en la figura 4.9. Por lo tanto la hebra de control cuando se presente este caso lo que deberá es indicar al robot que gire hacia la derecha para no perder la

pared.

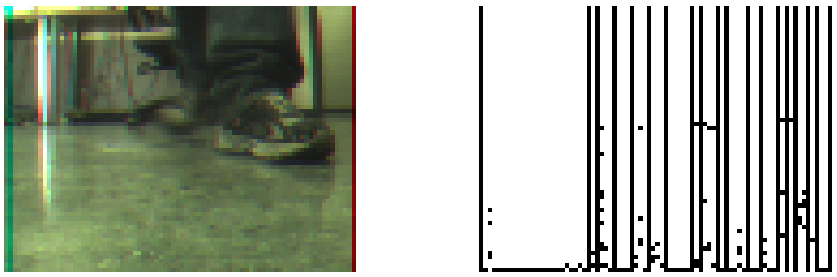

Figura 4.9: Visualización de los segmentos en el caso de discontinuidad.

- El caso en el que el robot ve pared infinita como en la figura 4.10. Es decir tiene reconocida la línea frontera y la observa con cierta orientación. Esta orientación se va a precisar mirando los valores que posee el punto de la derecha de los dos puntos que caracterizan al denominado *segmento guia*, los valores de la coordenada $x$ y de la coordenada $y$ para este punto, ya que la pendiente es menos fiable. De modo que si estos valores se corresponden con los capturados en la fase de calibración para que el robot pudiera obtener una referencia de a que distancia de la pared debía desplazarse, es que ve el *segmento guia* igual que en la fase de calibración y por lo tanto la hebra de control en ese caso indicará al robot que avance solo en linea recta ya que esta alineado con la pared.

Figura 4.10: Visualización de los segmentos en el caso de pared infinita.

En caso contrario de que no se correspondan con los valores capturados en la fase de calibración, existen dos situaciones posibles. El primer caso es que la orientación es distinta porque se esta alejando de la pared a seguir, en cuyo caso

el control ordena girar hacia la pared para acercarse. O por contra, un segundo caso, en el que la orientación es distinta porque se este acercando el robot a la pared, en cuyo caso la hebra de control indica al robot que gire hacia la izquierda. En estos dos casos la velocidad de giro va en función cuanto se esta desviando de la trayectoria correcta. Este porcentaje será la diferencia entre los valores de la calibración y los actuales, para el punto de la derecha del *segmento guia*.

- En el caso en el que el robot EyeBot visualiza una esquina cóncava, lo que realmente esta viendo al procesar la imagen, es que además del *segmento guia* que le sirve como referencia de su posición respecto de la pared, vera tras este otro segmento situado a la izquierda de este, justo a continuación como en la figura 4.11. De tal modo que si es una esquina cóncava los valores para el punto inicial del *segmento guia* serán parecidos o iguales los del punto final del segmento que esta a su izquierda, además de que este segmento tendrá una longitud considerable ya que será una pared frontal. En este caso la orden de control que se dictará es la de girar a la izquierda para evitar el choque con la pared frontal identificada. Pero el robot actuara de esta manera cuando este a una determinada distancia de la pared frontal, que se estima en función de la longitud del *segmento guia*. Cuando esta longitud es pequeña es porque se acaba la pared lateral y llega la frontal, así que es el momento de actuar girando a la izquierda.

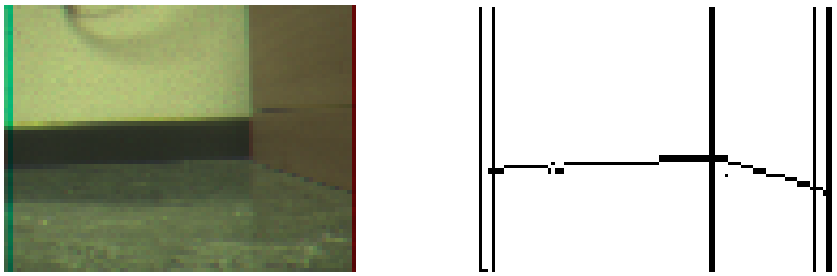

Figura 4.11: Visualización de los segmentos en el caso de esquina cóncava.

- Otro caso es cuando el robot EyeBot visualiza una esquina convexa, esto sucede cuando aparecen varios segmentos en la imagen además del *segmento guia*, de modo que el que esta más a su izquierda es longitud muy pequeña y los valores de sus puntos en el eje $Y$ son muy distintos, el valor $y$ del primer punto del

segmento, empezando por la izquierda, es mucho mayor que el valor del segundo. Cuando se produce este caso el robot visualiza una imagen como al de la figura 4.12, y el movimiento que deberá llevar a cabo será girar a la derecha cuando se le acabe la pared lateral, que es cuando la longitud del *segmento guia* sea demasiado pequeña.

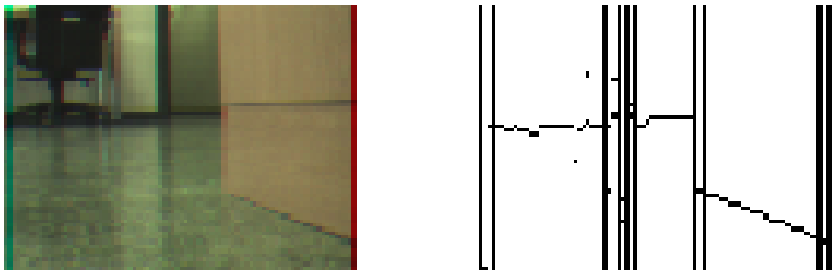

Figura 4.12: Visualización de los segmentos en el caso de esquina convexa.

# Capítulo 5

# CONCLUSIONES Y MEJORAS

En este apartado se citan las conclusiones más relevantes que se han extraído durante el tiempo que duró la generación del comportamiento sigue pared mediante visión. Así como las posibles mejoras que se pudieran introducir.

## 5.1. CONCLUSIONES

Como principal conclusión se puede extraer que ha sido posible implementar el comportamiento sigue pared mediante visión sobre la plataforma del robot EyeBot. Además se ha conseguido que este comportamiento se desarrolle en tiempo real, de modo que procesa de 2 a 3 imágenes por segundo, cuando la velocidad máxima si solo se capturan y no se realiza ningún tratamiento sobre ellas es entorno a las 3.7 imágenes por segundo.

El comportamiento se ha diseñado acorde con la filosofía JDE, de manera que el esquema perceptivo del comportamiento extrae la información relevante de las imágenes, que es la caracterización de la línea frontera entre el suelo y la pared. Y la tarea de control ordena al EyeBot cómo tiene que moverse, mediante el análisis de casos. De este modo se ha demostrado la validez de esta arquitectura de control JDE para el desarrollo de comportamientos autónomos en la plataforma del robot EyeBot.

En cuanto a la implementación, el diseño se ha materializado en un programa con tres hebras. El esquema perceptivo esta implementado como una hebra que realiza tres funciones: una realiza un filtro de color sobre las imágenes para discriminar el suelo, otra determina la linea frontera entre el suelo y la pared y por último una se encarga de segmentar esa línea. La función que realiza el filtro de color trabaja con las imágenes en

el espacio RGB, descartándose el espacio HSI ya que era computacionalmente más costoso. La extracción de la frontera se ha implementado utilizando un algoritmo robusto que realiza un barrido de las imágenes de abajo hacia arriba. La función que segmenta la línea frontera esta implementada como un *algoritmo abductivo* muy eficiente computacionalmente, lo que permite el procesado de las imágenes sea rápido, obteniéndose más rápidamente la información relevante para poder llevar a cabo la actuación. Lo que hace que se cumpla el requisito de que el comportamiento se desarrolle en tiempo real.

Por su parte el esquema de control es otra hebra que analiza los casos en los que se encuentra el robot y acciona los motores de este para moverlo en consecuencia. De manera que el robot pueda afrontar el mayor número de situaciones que se le presenten cuando desarrolle el comportamiento. Funcionando perfectamente para los casos de todo pared, todo suelo, esquina cóncava y pared infinita en el que se ha conseguido que las rectificaciones del robot sean pequeñas de modo que en su trayectoria el rizado sea leve.

Hemos identificado una serie de problemas relevantes. Uno de estos problemas es que la percepción debe ser lo más rápida posible a la hora de desarrollar el comportamiento, ya que si esto no es así el control actuara a destiempo con el consabido riesgo de que el robot se pierda o colisione. Este aspecto se ha resuelto implementando las funciones de tratamiento de imágenes de manera eficiente de modo que el procesado de las imágenes sea lo mas rápido posible. Esto se ha realizado evitando trabajar con reales (*float*), ya que al procesador le costaba más tiempo trabajar con estos, al no poseer coprocesador matématico. Otro solución que mejora este aspecto de la rapidez perceptiva es la de la visualización opcional de las imágenes que captura el robot en pantalla. Esta visualización es útil para la depuración del comportamiento, pero no a la hora llevarlo a cabo ya que ralentiza este. De modo que si no se visualizan la ganancia es de una imagen por segundo aproximadamente.

Una de las principales conclusiones extraídas de estos problemas es que para el desarrollo del comportamiento es crucial disponer de un buen sistema de percepción del entorno. De modo que este sea lo más rápido posible en la generación de la información relevante. En caso contrario la actuación se llevaría a cabo a destiempo y el robot no daría impresión de actuar de manera dinámica durante el desarrollo del comportamiento.

## 5.2. LINEAS FUTURAS

Uno de los problemas surgidos durante el desarrollo del comportamiento es que al trabajar con las imágenes en espacio RGB las condiciones de luminosidad afectan mucho a la percepción del robot. De modo que el comportamiento debe desarrollarse bajo condiciones de luz controladas. De manera que una posible línea futura de investigación será la de mejorar el filtro de color para que este sea lo más robusto posible y se desarrolle en condiciones variables de luminosidad.

Otra posible linea de investigación sobre este comportamiento, es la de mejorar el análisis de casos que se realiza en el esquema de control. De modo que el robot afronte todas las situaciones que le se le presenten con éxito. Ya que han quedado incompletos ciertos casos en los que se puede encontrar el robot. Es el caso del análisis de la esquina convexa, o la denominada *discontinuidad espacial*, ya que quedan algunas situaciones por estudiar que se podrían encapsular dentro de estos casos.

Por último otra posible línea futura es la de intentar fusionar los estímulos percibidos por la cámara con la información extraída por los sensores infrarrojos. Con el único fin de hacer este comportamiento mucho más completo.

# Bibliografía


[Agüero02] AGÜERO, C. E.: *"Protocolo de comunicaciones para el robot EyeBot."*, Proyecto Fin de Carrera, Universidad Rey Juan Carlos, 2002.

[Álvarez02] ALVAREZ, J. M.: *"Implementación basada en lógica borrosa de jugadores para la RoboCup."*, Proyecto Fin de Carrera, Universidad Rey Juan Carlos, 2002.

[Cañas02a] GARCÍA, E., CAÑAS, J. M., MATELLÁN, V.: *"Manual del Robot EyeBot"*, Informe Técnico del GSyc, 2002.

[Cañas02b] 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.

[GarcíaMorata02] GARCÍA, E.: *"Construcción de un teleoperador para el robot EyeBot."*, Proyecto Fin de Carrera, Universidad Carlos III, 2002.

[Kernighan99] KERNIGHAN, B. W., PIKE, R.: *"The Practice of Programming"*, Addison-Wesley, 1999.

[Kernighan91] KERNINGHAN, B. W., RITCHIE, D. M.: *"El Lenguaje de Programación C"*. Prentice Hall, 1991.

[Matellán00] MATELLÁN, V. CAÑAS, J. M.: *"Interacción persona robot hoy"*. Actas del III Congreso Interacción persona-ordenador, Universidad Carlos III de Madrid, Mayo 2002.

[Salinas97] SALINAS, B. C., SAORÍN, P. L., MIRA, J. M., RUIZ, A. P. Ruiz, GUILLÉN, S. S.: *"Latex"*, AD, 1997.

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

[Strelb00] STRELB, M. D., TURNER, M.: *Linux*, Prentice Hall, 2000.