UNIVERSIDAD
REY JUAN CARLOS

INGENIERÍA INFORMÁTICA

Curso académico 2005-2006

Proyecto Fin de Carrera

NAVEGACIÓN DE UN ROBOT CON UN SISTEMA DE ATENCIÓN VISUAL 3D

Autor: Olmo León Cadahía
Tutores: José María Cañas Plaza
Pablo Barrera González

A Marta y Paula, siempre juntos.

A Dolo, siempre juntos.

. . .

A mi Angel, siempre juntos.

Agradecimentos.

Quiero agradecer la compañía y ayuda prestada durante la realización de este proyecto a Víctor Gómez, Carlos Agüero, Ricardo Palacios, Antonio Pineda y especialmente a Pablo Barrera.

Mis más sinceros agradecimientos a José María Cañas, que ha sabido guiarme perfecta y pacientemente, contagiándome también su energía y ganas de investigar.

Índice general

Índice de figuras

Índice de cuadros

Resumen

Los sistemas de visión son hoy en día uno de los elementos sensoriales más atractivos para incrementar la autonomía en robótica. Las cámaras proveen mucha información sobre el entorno del robot y son unos sensores bastante baratos. Pero tienen una desventaja. Extraer información a partir de las imágenes capturadas por una cámara es algo difícil, en parte porque hay que distinguir aquello que es importante dentro de las imágenes de lo que no lo es. Para ello están los sistemas de atención visual.

El presente proyecto realiza un algoritmo de atención visual tridimensional, para que un robot sea capaz de navegar autónomamente. Para ello realiza un control de atención sobre los objetos relevantes que aparecen en el *mundo 3D* que rodea al robot, manteniendo una representación de la escena. Para poder extraer la información tridimensional de los objetos se cuenta con un par estéreo de cámaras. Como la escena abarca mucho más de lo que contienen las imágenes estáticas del par estéreo, se ha empleado un cuello mecánico para moverlas, que se sitúa sobre el robot.

El algoritmo mueve de manera inteligente este cuello con el objetivo de encontrar objetos relevantes, como balizas o la frontera del suelo, ver y afianzar los ya conocidos y olvidar aquellos objetos que ya no aparezcan en la escena. Al mismo tiempo, analiza su posición tridimensional, manteniendo una representación de la escena 3D. Por último, usa esta información del mundo para poder navegar por el autónomamente, siguiendo balizas y evitando chocar con obstáculos, únicamente mediante visión.

Para su implementación, se ha empleado la plataforma software *jde.c* y se han programado una serie de esquemas escritos en C. Concretamente se han implementado dos nuevos esquemas. El más importante es el encargado de los cálculos geométricos necesarios para la obtención de la información tridimensional. También se han integrado otros esquemas para adaptarlos a las nuevas necesidades de este proyecto.

Capítulo 1

Introducción

En este capítulo se va a dar una visión panorámica del contexto del presente proyecto. Para ello se comienza dando una visión general del mismo, describiendo qué es la robótica, ya que se ha usado un robot, en qué consiste la navegación de robots y la visión artificial en robots, debido a que se ha conseguido que el robot navegue con la ayuda de cámaras. Finalmente, se hace hincapié en el contexto más próximo en el que se centra, que son los sistemas de atención y la localización visual de objetos en 3D.

1.1. Robótica

La palabra robot deriva de la palabra checoslovaca *robota*, que significa trabajador, sirviente. Surge en la obra RUR, los *"Robots Universales de Rossum"*, escrita por Karel Capek (1921). En esta obra se plantea la construcción de robots para liberar a las personas de la pesada carga del trabajo. A su vez, el escritor Isaac Asimov, a quién se atribuye el término robótica, previó un mundo futuro en el que existían reglas de seguridad para que los robots no pudieran ser dañinos para los seres humanos. Siguiendo en esta línea siempre han existido novelas de ciencia-ficción en las que se creaban seres artificiales a imagen y semejanza de los seres humanos. Ejemplos de estas novelas son *Frankenstein*, la vieja leyenda judía del *Golem*, etc. Pero todo esto se queda en ficción. En la realidad fue en 1954 cuando George Devol desarrolló un brazo primitivo o manipulador que sería el origen real de los robots industriales.

Los robots industriales (figura 1.1) fueron evolucionando y tienen su apogeo hoy día. Gracias a ellos se dio lugar a unos procesos de producción mucho más eficientes y aportaron mayor calidad a los productos. Estos robots suelen ser empleados en tareas repetitivas y pesadas, como pueden ser el soldado, el pintado o la carga de

maquinaria. Suelen estar pre-programados para esa tarea y no disponen de capacidad para reconfigurarse autónomamente.

Figura 1.1: Ejemplo de un robot industrial

A finales de los años 70 y a principios de los años 80, surgió una nueva aproximación en la robótica en el mundo de la investigación. Este nuevo tipo de robótica se denomina *Robótica Autónoma*. Los robots autónomos son sistemas que operan en entornos complejos sin necesidad de estar constantemente guiados y controlados por operadores humanos. Una propiedad fundamental que poseen es la de poder reaccionar adecuadamente ante situaciones no previstas explícitamente en su programación previa. Por ello, se suele considerar a la robótica autónoma un campo muy relacionado con la inteligencia artificial, puesto que para generar un comportamiento adecuado se requiere de una cierta inteligencia mínima. Un robot de este tipo dispondrá de una serie de sensores, actuadores, y computadores. Estos últimos hay que programarlos para dotar de *inteligencia* al robot, por tanto, su inteligencia reside en su software. Hoy día, sigue siendo un reto dotar a los robots autónomos de una inteligencia elevada.

Actualmente, los robots autónomos tienen diversas aplicaciones. Una muy importante es el uso que se les da en tareas peligrosas o desagradables para los seres humanos, como pueden ser exploraciones submarinas, volcánicas o espaciales. Ejemplo de ellas la realizada por los robots espaciales Spirit y Opportunity de la NASA (figura 1.2), que se posaron en Marte (2004) y realizaron tareas como la detección de agua subterránea en dicho planeta.

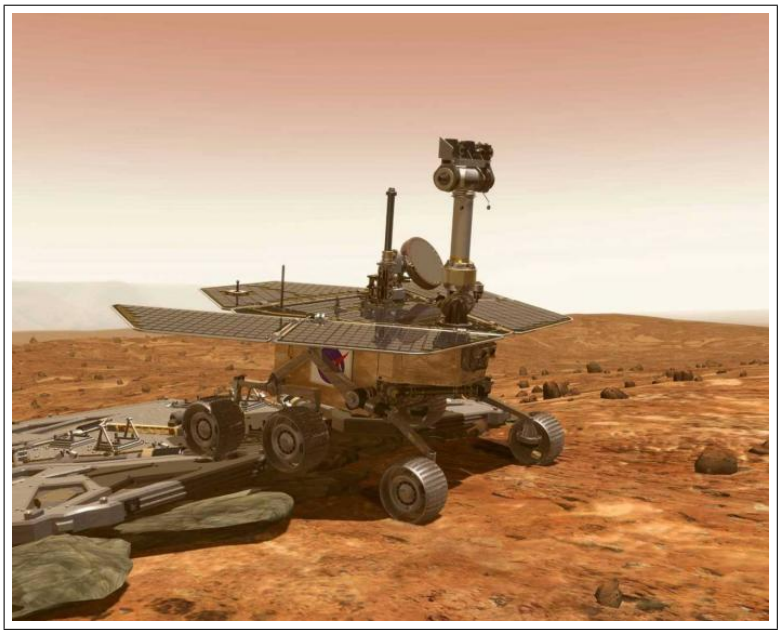

Figura 1.2: Robot espacial Opportunity

Actualmente se ha logrado un gran avance en los robots dedicados a la medicina, y es que se han empleado robots teleoperados en procedimientos de cirugía, ya que la precisión de movimiento de un robot es superior a la de las manos de un cirujano, inevitablemente sujetas a movimientos no deseados o a errores por falta de visibilidad, cansancio, etc. También se usan robots dentro de los laboratorios para transportar muestras biológicas o químicas entre instrumentos tales como incubadoras, manejadores de líquidos y lectores.

Otra aplicación a destacar es la *robótica de servicio*. Los robots pueden colaborar dentro de nuestras casa y oficinas en multitud de tareas, que van desde el simple ocio (robots mascotas, como el perrito Aibo de Sony (figura 1.3 izquierda)), pasando por el desarrollo de tareas domésticas, como una cortadora de césped o la aspiradora Roomba (que ha vendido 1,5 millones de ejemplares (figura 1.3 derecha)), hasta tareas de vigilancia y seguridad. El grado de aceptación de estos robots sigue creciendo, aunque su comercialización está limitada por su elevado coste.

El presente proyecto se enmarca dentro de la robótica autónoma, ya que se ha programado la inteligencia del robot para que, gracias a los sensores de los que dispone, sea capaz de navegar autónomamente.

Figura 1.3: El robot aibo de Sony (Izquierda) y el robot aspiradora roomba (Derecha).

1.2. Navegación de robots

El primer problema de un robot es cómo moverse por su entorno. Llamamos navegación a la finalidad de conducir un robot móvil mientras atraviesa un entorno, para alcanzar un destino o meta, sin chocar con ningún obstáculo, ya sea estático o dinámico. El robot de este proyecto fin de carrera tendrá que navegar autónomamente siguiendo balizas y evitando chocar con obstáculos y parédes.

Tradicionalmente este problema se ha dividido en dos tipos de navegación, por un lado la navegación global donde se dispone de un mapa del entorno y por otro lado la navegación local donde el robot conoce el entorno mediante sus sensores.

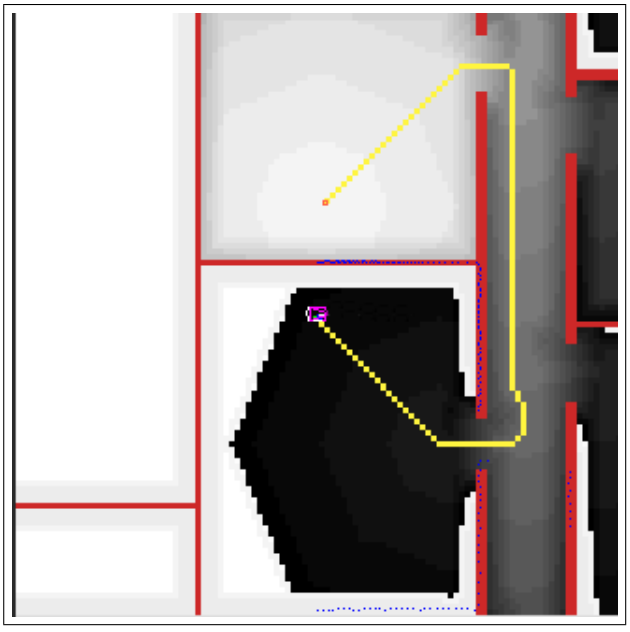

Figura 1.4: Navegación global basada en gradiente

La navegación global se basa en calcular la ruta antes de empezar a moverse. Para ello se tiene que conocer el entorno de antemano, bien con un mapa conocido a priori, o bien generando un mapa del entorno con los sensores del robot. Una vez que el robot tiene conocimiento del entorno y sabe dónde se encuentra en ese mapa, se calcula la ruta a seguir. Este tipo de navegación tiene el inconveniente de los obstáculos dinámicos, es decir, cuando el robot va hacia su destino si le aparece un objeto que no tiene en el mapa (por ejemplo una persona), el robot no sabría evitarlo. Algunas técnicas desarrolladas en este tipo de navegación son: *Grafo de visibilidad* (Nilson 1969) [López, 2005], *Diagrámas de Voronoi* y *planificación basada en gradiente* (Konolige, 2000). En la figura 1.4 podemos ver un ejemplo de navegación global [Isado, 2005] utilizando la planificación basada en gradiente.

La *navegación local*, es reactiva ya que basándose en las medidas obtenidas del entorno en *cada instante* y sin utilizar mapas, el robot toma una decisión en ese mismo momento. Así con este tipo de navegación podemos seguir algo, ir en una dirección determinada, pero no podemos ir de una parte de un edificio a otra al no tener conocimiento a priori de un mapa. Dado que en este proyecto se ha usado la navegación local comentaremos algunas técnicas.

- Método de velocidades y curvaturas (CVM): En este método los obstáculos se modelan por círculos. El objetivo es encontrar la mejor trayectoria, que será aquella que más se aleje de los obstáculos. El robot se moverá en trayectorias circulares y su velocidad estará restringida. Las restricciones derivan de las limitaciones físicas de las velocidades, aceleración propias del robot, y de los datos sensoriales que indican la presencia de obstáculos.

- Método de carriles y velocidad(LVM): Este método combina el *CVM* con un nuevo método llamado "de carriles". Este divide el entorno en carriles, y luego elige el mejor a seguir hacia la dirección deseada. Así se consigue un movimiento libre de colisiones con un movimiento suave teniendo en cuenta la dinámica del robot.

- Campos de fuerzas virtuales (VFF) (J.Borenstein, 1989): Con esta técnica el robot tiene que viajar hacia un objetivo, que ejerce una fuerza atractiva sobre él, simulando un tirón gravitatorio. Sin embargo no es ésta la única fuerza que actúa sobre el robot. Puesto que el fin último de esta técnica es evitar los obstáculos más cercanos al robot, éstos generan una fuerza repulsiva sobre el robot, mayor

cuanto más cerca está de éste. El movimiento del robot se ve gobernado por la suma vectorial de todas la fuerzas que afectan al robot. Con este métedo se logra un compromiso entre la tendencia a alejarse de los obstáculos y la tendencia de llegar al objetivo deseado. Esta técnica es la que se ha empleado para lograr los objetivos de este proyecto.

En aplicaciones reales se usan técnicas de *navegación híbrida* que combina la planificación de caminos de la *navegación global* con la ejecución reactiva de la *navegación local*. Algunos ejemplos de esta navegación son: el robot *XAVIER* en CMU y el vehículo *Stanley* ganador de la carrera *Darpa Grand Challange-2005*, que se realiza cada año y donde vehículos no tripulados han de realizar un recorrido de forma completamente autónoma.

Siguiendo esta línea se encuentra el presente proyecto, donde los únicos sensores que se usan para conocer el entorno y así poder navegar, son las cámaras. Por esto, se da paso a la sección de visión artificial.

1.3. Visión Artificial en Robótica

La visión artificial o visión por computador, es una rama de la inteligencia artificial que tiene por objetivo analizar y proponer herramientas para el procesamiento de imágenes, reconocimiento de patrones (como pueden ser caras), reconstrucción de mapas 3D, etc . Además, es una ciencia que posee numerosas aplicaciones en el mundo real, como puede ser dentro de la medicina, la industria, la robótica, etc.

La visión artificial surge en la década de los 60 con la idea básica de conectar una cámara de vídeo a una computadora. Esto implicó no sólo la captura de imágenes a través de la cámara, sino también la comprensión de lo que estas imágenes representaban. Un resultado muy importante de este trabajo y que marcó el inicio de la visión artificial, fue un trabajo realizado por Larry Roberts, el creador de *ARPAnet*. En 1961 creó un programa, el *"mundo de micro-bloques"*, en el que un robot podía "ver" una estructura de bloques sobre una mesa, analizar su contenido y reproducirla desde otra perspectiva, demostrando así que esa información visual que había sido mandada al ordenador por una cámara, había sido procesada adecuadamente por él. Sus herramientas están agrupadas por el procesamiento digital de imágenes y por el reconocimiento de patrones.

Sin embargo, hasta comienzos de la década de 1990 no comenzaron a aparecer ordenadores con velocidad de cómputo suficiente para procesar imágenes de forma ágil y con poca demora. A partir de entonces la visión computacional comenzó a emplearse para múltiples tareas y su desarrollo fue concentrándose en problemas de tratamiento de imágenes como la segmentación, el reconocimiento de formas o el filtrado de bordes.

Mediante el uso de estas técnicas fue posible alcanzar numerosos propósitos tales como: *reconocimientos faciales* (figura 1.5 (a)), *reconocimiento de objetos* (figura 1.5 (b)), empleado a menudo en la industria para ayudar en el control de calidad, *seguimiento de objetos 2D* o la *detección de movimiento*, usados frecuentemente en entornos de seguridad para el control de personas.

Figura 1.5: Reconocimiento con visión artificial - Facial (a) y Objeto (b).

En la actualidad, los avances en geometría proyectiva, pares estéreo y autocalibración han abierto la puerta a la extracción de *información tridimensional* de las imágenes. El uso de estos nuevos datos permite alcanzar nuevas cotas en el área de la visión artificial, pues la información tridimensional permite conocer mejor y con una mayor precisión el entorno.

Gracias al impulso de estas nuevas técnicas es posible hoy día realizar *reconocimiento 3D* de rostros u objetos. Estos pueden ser utilizados en distintos ámbitos como la representación de piezas en entornos industriales, el reconocimiento de personas mediante el estudio biométrico en áreas de seguridad (figura 1.6), diseño infográfico para la industria del cine o de videojuegos, etc.

Otra de las aplicaciones de la visión artificial con respecto a la extracción de

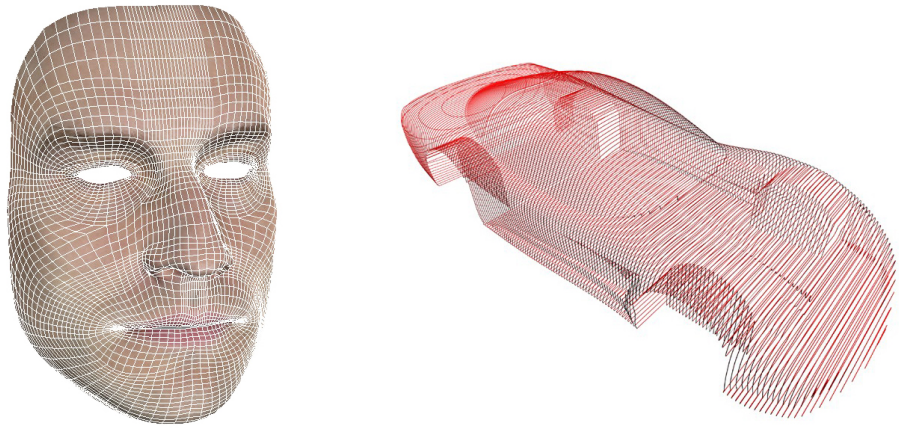

Figura 1.6: Modelado tridimensional - Rostro humano (a) y Pieza industrial (b)

información 3D es la que utiliza el sistema de *repetición tridimensional de jugadas deportivas* Eye-Vision[1]. A través de la colocación estratégica de un determinado número de cámaras alrededor de un estadio es posible obtener la realidad virtualizada de lo sucedido segundos antes. La escena tridimensional se obtiene de la composición de las imágenes 2D que consigue cada una de las cámaras.

Figura 1.7: Sistema de repetición tridimensional Eye-Vision.

La visión en la robótica puede jugar un papel fundamental. Actúa como un sensor más, pero es potencialmente muy rico y a la vez muy barato. Algunos ejemplos de lo desarrollado por el grupo de robótica de la URJC en este campo son: *Seguimiento de un objeto en 3D mediante un filtro de Partículas* [Barrera, 2005], y *Reconstrucción 3D mediante un algoritmo evolutivo* [Cañas, 2005]. En este proyecto, de igual modo, se han utilizado dos cámaras con el objetivo de poder localizar mediante visión la posición 3D de diversos objetos.

[1] http://www.ri.cmu.edu/events/sb35/tksuperbowl.html

1.4. Localización Visual de Objetos

En términos generales se debe entender por localización al proceso de calcular *la posición de un objeto* dentro de un entorno. El conocimiento de esta posición viene determinado por la información que se obtiene de los sensores del sistema.

Mediante la visión artificial se puede obtener una representación del entorno a base de localizar los diversos objetos que rodean las cámaras. Esta reconstrucción de la escena puede hacerse de varias formas según el objetivo deseado. En un extremo existe la *reconstrucción densa*, donde se localizan todos los puntos de la escena [Bustos, 2003], para tener una representación del entorno completa. Y en el otro extremo está la de encontrar los objetos interesantes de la escena, y después obtener su localización. Un ejemplo de éste segundo método es el utilizado por los perros Aibo (dotados de una sola cámara como sensor principal) para competir en la *Robocup*, donde lo que interesa es saber dónde está la pelota, la portería y los otros jugadores. Éste método de localización es el más adecuado si se desea una velocidad de cómputo lo suficientemente rápida como para hacer navegar un robot en tiempo real, ya que el resto de información no es relevante para la tarea. Éste es el sistema utilizado en este proyecto.

La localización 3D consiste en determinar la *posición de un objeto* o elemento dentro de un *entorno tridimensional*. Esta posición habitualmente vendrá expresada como un punto tridimensional de coordenadas (x,y,z), cuya situación estará referida a un orígen o centro de referencia previamente establecido.

Para lograr la localización 3D es necesario el uso de cámaras u otros sensores que informen de la posición del objeto. En el caso de las cámaras, se necesitan al menos dos imágenes de posiciones diferentes apuntando al mismo objeto para poder conseguir la localización 3D. Mediante las dos imágenes es posible aplicar una técnica de localización que determine la posición. Ejemplos de este uso están en los proyectos *Localización 3D* [Pineda, 2006] una aplicación para detectar la posición 3D de una persona en una habitación mediante cámaras estáticas, y *Escena Rica* [Palacios, 2006] donde un robot pioneer navega y reconstruye la escena con la ayuda de sensores como cámaras y un detector láser; proyectos realizados dentro del grupo de robótica de la URJC [2].

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

Una de las aplicaciones más conocidas del seguimiento tridimensional son los sistemas Vicon[3] de captura de movimiento (figura 1.8). Estos sistemas son ampliamente utilizados en la industria cinematográfica y de videojuegos para la captura de los movimientos de personas, que serán aplicados a personajes virtuales generados por ordenador. Esta técnica se basa en la grabación con múltiples cámaras de una persona vestida con un traje especial. Este traje lleva adheridos ciertos elementos que reflejan muy bien luz infrarroja emitida por gran cantidad de diodos situados en cada cámara y que es captada por las mismas desde distintos puntos de observación. Con esta información, un ordenador puede generar una representación en movimiento de los puntos del traje, que puede ser aplicada a un modelo tridimensional para animarlo.

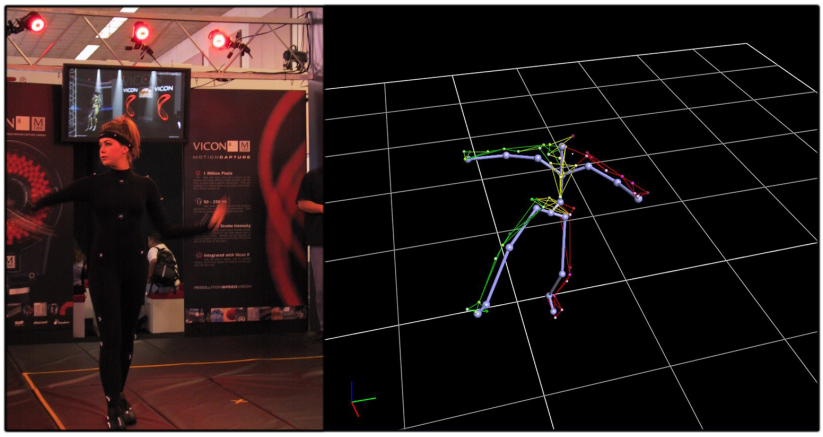

Figura 1.8: Sistema de captura de movimiento de Vicon Peak.

1.5. Navegación con Atención Visual

Dentro de la visión artificial se encuentra la atención visual. La atención es la capacidad de fijarse en uno o varios aspectos de la realidad, prescindiendo de los restantes. Esto es, tener preferencia sobre ciertos objetos que sobresalen por alguna característica, ya sea por su color, forma, etc. De esta forma, se pueden gastar los recursos disponibles solo en las cosas que interesan.

Dentro de la robótica autónoma es importante realizar un control de atención visual. Las cámaras de los robots proveen de un amplio flujo de datos del que hay que seleccionar lo que es interesante e ignorar lo que no. En esto consiste la atención visual selectiva. Existen dos vertientes de atención visual, *global* (*overt attention*) y

[3]http://www.vicon.com/

local (*covert attention*). La atención *local* consiste en seleccionar dentro de una sola imagen aquellos datos que nos interesan. Y la atención *global* consiste en seleccionar del entorno que rodea al robot aquellos objetos que interesan, a los que dirigir la mirada [Cañas, 2005].

Un ejemplo de este uso está en el proyecto *Sistema de atención visual en escena* [Martínez, 2005]. Este proyecto estaba enfocado a realizar un control de atención sobre diferentes objetos situados en una escena visual, cuyo campo de visión abarcaba más que el propio de una sola cámara. Los objetos a atender eran relevantes por su color, se identificaban y se hacía un seguimiento entre ellos, olvidándolos cuando desaparecían de la escena. En este proyecto el robot no navegaba, y se disponía de una sola cámara, por tanto, no se hacía localización 3D.

En otros proyectos de este grupo se ha navegado con el robot usando visión: *Comportamiento sigue persona con visión direccional* [Calvo, 2004], en el que se disponía de un sensor láser que ayudaba a la navegación; o *Navegación visual del robot pioneer* [Diaz, 2005] en el que se consiguió navegar autónomamente mediante visión, pero con un sistema de atención pasivo, en el que no se sabe cuántos objetos interesentes hay en la escena, ni en qué posición se encuentran.

Siguiendo la línea de la atención visual, de la navegación y de la localización 3D, se plantea el presente proyecto. Este proyecto está enfocado en realizar un control de atención sobre diferentes objetos situados en *el mundo* y estimar la posición tridimensional de dichos objetos para obtener una representación de ese *mundo*. Por último se usa la información obtenida para que el robot sea capaz de navegar autónomamente, solo mediante visión. De esta forma, pondrá su atención sobre diferentes balizas repartidas en la escena, que deberá buscar e ir siguiendo. Al mismo tiempo, deberá prestar atención a los bordes del suelo para reconstruir los límites de su entorno y así evitar chocar.

Esta memoria consta de seis capítulos. En el segundo capítulo se describirá cuál es el objetivo concreto del proyecto y qué requisitos, funcionales y no funcionales, se deben cumplir para un correcto funcionamiento del mismo. En el tercero se describirán las herramientas necesarias para su implementación, tanto herramientas hardware como software. En el capítulo cuatro se explicará cómo se ha desarrollado el sistema, hablando tanto del diseño como de la programación del mismo. En el capítulo cinco se comentarán

las pruebas experimentales realizadas, para observar cómo se han cumplido los objetivos y ver el correcto funcionamiento del sistema. Finalmente, se presentará en el capítulo seis las conclusiones que se han obtenido y qué mejoras se pueden efectuar para posibles trabajos futuros.

Capítulo 2

Objetivos

Tras haber presentado el contexto general y particular sobre el que se asienta la aplicación, estableceremos en este capítulo los objetivos concretos que se han propuesto resolver mediante este proyecto fin de carrera al igual que los requisitos que han condicionado el desarrollo del mismo.

2.1. Descripción del Problema

El objetivo del proyecto consiste en intentar mantener una representación de los objetos interesantes del *mundo 3D* que rodea al robot, mediante un algoritmo de atención visual. La aplicación desarrollada deberá ser capaz de explorar el entorno del robot, reconstruyendo los límites del suelo que encuentre, y buscando una serie de balizas. Después deberá navegar autónomamente en dirección a la baliza más cercana que haya visto y evitando chocar con las paredes. Cuando alcance su objetivo, comenzará a buscar balizas de un tipo distinto a la primera, diferenciadas por los colores. Todo este comportamiento estará respaldado por una representación de la escena tridimensional que irá manteniendo dinámica y atentivamente. Para ello se contará, como sensores principales, con dos cámaras, de las que se deberá obtener toda la información del entorno, incluyendo la localización tridimensional de los *objetos relevantes*.
Dicho objetivo se ha articulado en cuatro subobjetivos más específicos:

1. *"Procesamiento de Imágenes Monoculares"*. La cámara deberá ir capturando imágenes continuamente para conocer la información del entorno en el que está situada.

   a) *"Capturar Imágenes"*. El sistema deberá ser capaz de realizar una toma de

imágenes mediante cámaras firewire.

b) *"Distinguir objetos relevantes de los no relevantes en una imagen"*. Las imágenes capturadas deberán ser analizadas mediante técnicas propias de visión computacional: filtrado de color, filtrado de bordes y segmentación 2D, para identificar los objetos relevantes (las balizas y la frontera del suelo).

2. *"Localización 3D de Objetos"*. La aplicación debe ser capaz de estimar la posición tridimensional de objetos, utilizando únicamente los datos obtenidos de las dos cámaras, mediante técnicas geométricas. Deberá emparejar correctamente los objetos vistos en cada cámara y triangular desde cada una para calcular su posición 3D. A su vez ha de mantener una representación visual de la escena en una imagen virtual, en la que las balizas tendrán asociadas un punto 3D (x,y,z) y los bordes del suelo reconocidos serán segmentos 3D (*punto 3D inicial, punto3D final*.

3. *"Control de Atención"*. Se deberá realizar un control de atención sobre los diferentes objetos identificados en el *mundo 3D*. Para ampliar el rango de visión a todo lo que hay alrededor del robot, habrá que mover las cámaras, y se utilizará para ello un cuello mecánico. Las características deseables para este sistema de atención serán:

   a) *"Reconocer nuevos objetos"*. Cuando se analice una imagen y se descubra un objeto no identificado previamente, se deberá memorizar su posición para volver a ser visitado.

   b) *"Realizar un seguimiento sobre los diferentes objetos encontrados"*. Los diferentes objetos encontrados deberán ser atendidos periódicamente, es decir, deberán ser visitados de vez en cuando para poder realizar un seguimiento sobre ellos. Cuando un objeto previamente existente desaparezca de la escena, deberá ser olvidado.

   c) *"Explorar nuevas zonas en busca de objetos"*. Se deberá explorar zonas de la escena con la idea de captar nuevos objetos.

   d) *"Balizas objetivo"*. El algoritmo de atención deberá encontrar una serie de balizas visuales, y se fijará la más cercana al robot como objetivo. Por consiguiente se deberá saber con exactitud dónde está cada una en el mundo.

   e) *"Límites del suelo"*. También se incorporará como parte interesante de la escena los límites del suelo. Se deberá conocer con precisión dónde están y

hasta donde llegan.

4. *"Navegación Visual Autónoma"*. El robot pioneer deberá poder seguir las *balizas-objetivo* a la vez que evita chocar con las paredes, con la única ayuda de las cámaras como sensor de entorno. Si no se encontrasen balizas deberá ser capaz de deambular buscando zonas nuevas inexploradas. Cuando se alcance el objetivo deberá comenzar a explorar de nuevo, pero esta vez en busca de balizas con colores distintos a la anterior alcanzada.

El objetivo de este proyecto es que la aplicación funcione en un entorno controlado, siendo éste los pasillos del departamental de la ESCET. Como todo proyecto de ingeniería, se deberán realizar e incluir en esta memoria, diversos experimentos que prueben el funcionamiento de la aplicación.

2.2. Requisitos

En la siguiente sección se presentan los requisitos que deberán ser satisfechos por el sistema y los condicionantes que influyen en la solución elegida.

1. El lenguaje de programación empleado para la implementación del sistema será **ansi-C.**

2. Utilización del sistema operativo GNU/Linux.

3. El sistema será diseñado de acuerdo a la arquitectura JDE y su implementación software actual *jde.c* [Cañas, 2004]. Se utilizará esta arquitectura porque, además de que es la propuesta por el grupo de robótica, soluciona muchos problemas como la captura de imágenes, la configuración de los drivers, etc. Por lo tanto, siguiendo esta arquitectura se dividirá la solución del problema en esquemas.

4. El código implementado será software libre, por lo que se podrá utilizar y/o modificar de acuerdo con la licencia GPL (General Public License), publicada por la Fundación de Software Libre.

5. El sistema deberá ser lo más robusto posible, especialmente tolerante frente a cambios de iluminación en el entorno.

6. La captura de imágenes y su posterior tratamiento ha de ser en tiempo real (a 25 fps), por ello se deberán implementar algoritmos rápidos y eficientes.

7. Las cámaras deberán ser calibradas con precisión, para conocer sus parámetros y así poder realizar los cálculos geométricos necesarios.

8. La navegación deberá ser vivaz, por lo que los algoritmos a desarrollar deben ser eficientes.

2.3. Metodología

El plan de trabajo utilizado en la realización de este proyecto ha consistido en el *modelo de desarrollo en espiral basado en prototipos* (figura 2.1). Este modelo de desarrollo se basa en la realización de varias subtareas sencillas que conjuntamente compondrán el comportamiento final del sistema. Usando este modelo se aporta cierta flexibilidad en cuanto a posibles cambios de requisitos.

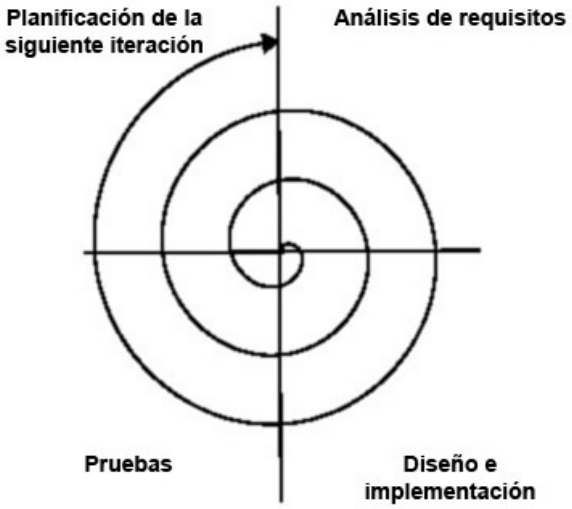

Figura 2.1: Modelo en espiral

Este modelo de desarrollo se caracteriza por la realización de las subtareas en un número determinado de ciclos. En cada uno de estos ciclos existen cuatro etapas: *Análisis de requisitos, Diseño e implementación, Pruebas y Planificación del próximo ciclo de desarrollo.*

Durante todo el desarrollo del proyecto se mantendrán reuniones semanales con los tutores para establecer y afinar los puntos llevados a cabo en cada etapa, y para comentar los resultados de etapas anteriores.

Para la consecución de este proyecto se deberán completar los siguientes ciclos:

1. *Formación y familiarización con la infraestructura:*

   a) *Familiarización con la plataforma software jde.c* y estudio a fondo de la estructura de la misma. La primera iteración consistirá en la creación de varios esquemas para comprender el funcionamiento general de *jde.c*.

   b) *Estudio de técnicas de visión artificial y otros conocimientos relativos* tales como filtros de color, segmentación, calibración de cámaras y técnicas de triangulación tridimensional. Diseño y creación de nuevos esquemas en los que se apliquen dichas técnicas.

   c) *Familiarización con el hardware* para comprenderlo y saber utilizarlo. Para ello se realizarán pruebas y programas sencillos con el cuello mecánico y el robot. Por último se implementará un nuevo esquema que simule el cuello, para ser usado en las pruebas en un entorno simulado controlado.

   d) *Estudio de otros esquemas*, como el de atención 2D [Martínez, 2005], con el fin de poder integrarlos y adaptarlos a las nuevas necesidades de este proyecto.

2. *Diseño e implementación del esquema de atención 3D*, para que funcione con localización y triangulación de objetos en tres dimensiones. Se integrarán también en esta etapa las técnicas de visión implementadas previamente, así como el esquema del cuello virtual. Por último, se dotará a la aplicación de una interfaz gráfica propia.

3. *Estudio e integración* de algoritmos de filtrado de bordes en imágenes, filtrado de líneas y de segmentos. Con esto se conseguirá obtener información acerca de los límites del suelo. En el último paso habrá que aplicar técnicas geométricas para obtener información de los bordes en 3D, e integrarlos como puntos de interés en el algoritmo de atención.

4. *Estudio de técnicas de navegación* y desarrollo de un algoritmo basado en campos de fuerza virtuales (VFF), donde la baliza sea la fuerza atractiva, y los bordes del suelo ejerzan fuerzas repulsivas.

5. *Experimentos y ajustes*. La última etapa de la espiral consistirá en experimentos y pruebas de integración realizadas con el robot real, realimentando la implementación persiguiendo la mejora.

2.4. Planificación Temporal del Proyecto

Para finalizar este capítulo se detalla la planificación del proyecto. Para ello, se enumeran las fases principales para la realización del sistema, y las tareas en las que se divide, así como las fechas límite de entrega para las tareas principales (Tabla 2.1).

Tarea	Nombre	Fecha Límite	Predecesoras
1	Preparación	dom 30/10/05	
1.1	Leer Documentación		
2	Visión	vie 25/11/05	1
2.1	Filtro HSI		
2.2	Segmentación Recursiva		2.1
2.3	Identificación de Objetos		2.2
2.4	Triangulación 3D		2.3
2.5	Calibración de Cámaras		
3	Familiarización Hardware	vie 09/12/05	1
3.1	Cuello Mecánico (pantilt)		
3.2	Sensores y Actuadores del Pioneer		
3.3	Esquema Simulador Pantilt		3.1
4	Estudio Algoritmo Atención 2D	vie 16/12/05	
5	Algoritmo Atención 3D	vie 14/04/06	2,3,4
5.1	Posiciones de objetos en 3D		
5.2	Resolución Ambigüedades entre Imágenes		5.1
5.3	Resolución Problema Observaciones Parciales		5.2
5.4	Representación Escena en Imagen Virtual		5.3
5.5	Integración Movimiento del Robot en Simulación		5.6
6	Realización Interfaz Gráfica	vie 15/08/06	2,3
7	Integración Líneas Suelo	mar 08/08/06	2,5
7.1	Filtro de Bordes, de Líneas y de Segmentos		
7.2	Triangulación 3D de Segmentos		7.1
7.3	Integrar Segmentos en Algoritmo Atención		7.2
8	Navegación VFF	jue 18/08/06	7
8.1	Navegar hacia Baliza Objetivo		
8.2	Navegar sin Chocar con Pared		8.1
9	Pruebas y Experimentos	mié 06/09/06	8
10	Documentación	vie 08/09/06	2,3,5,6,7,8,9

Cuadro 2.1: División en tareas del proyecto

Una vez descritos los objetivos, los requisitos, la metodología y la planificación de este proyecto, en el siguiente capitulo se analizará la infraestructura software y hardware sobre la que se asienta la aplicación.

Capítulo 3

Plataforma de Desarrollo

Tras haberse detallado los objetivos y requisitos, en este capítulo se describe la plataforma de desarrollo empleada, tanto hardware como software. Se comenzará dando una visión general de la misma y a continuación se describirá cada una de las partes, como son el robot *Pioneer*, la arquitectura software *JDE*, las cámaras digitales empleadas y el cuello mecánico *Pan-Tilt* entre otros.

La plataforma para realizar este proyecto es la que se muestra, a grandes rasgos, en la figura 3.1:

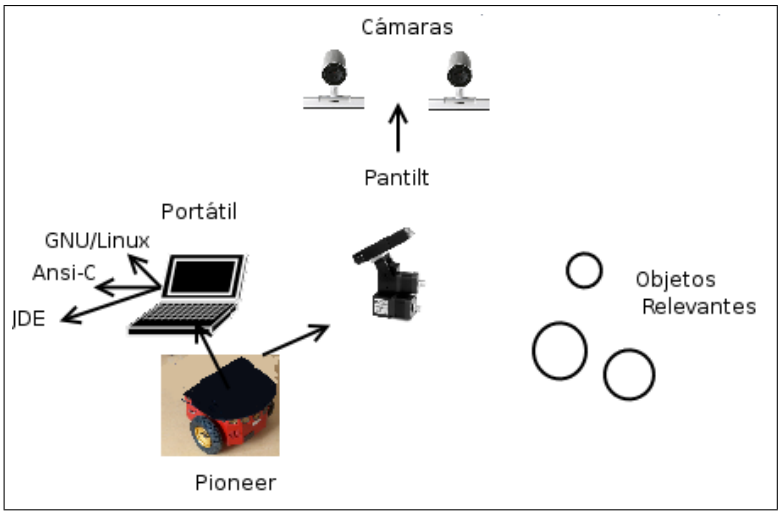

Figura 3.1: Plataforma empleada

Se tiene un robot *Pioneer* sobre el que está situado un ordenador portátil y un cuello mecánico. En el ordenador portátil está instalado el sistema operativo *GNU/Linux* y sobre él se ejecuta la plataforma software *JDE*. Ésta plataforma está basada en modelos llamados esquemas, escritos en *C*, por lo que el código está también escrito en dicho lenguaje de programación.

En cuanto al cuello mecánico, se ha instalado una tabla sobre él, para poder albergar las dos cámaras firewire encargadas de capturar las imágenes. Esta unidad permite un movimiento amplio horizontal y vertical, gracias al cual se puede llegar a ver una región mayor en un breve espacio de tiempo.

3.1. Plataforma Hardware

A continuación se presenta con mas detalle la plataforma hardware sobre la que se asienta la aplicación. En la figura 3.2 puede verse el robot con los elementos hardware añadidos a su base.

Figura 3.2: Robot Pioneer

3.1.1. El Robot Pioneer

Como se ha explicado, el robot real sobre el que funciona el sistema desarrollado en este proyecto es el robot *Pioneer2DXE*[1] [Cañas, 2004] (Figura 3.2). Este robot dispone de un equipo de elementos sensoriales para obtener información de su entorno, unos procesadores para realizar cálculos y unos motores para desplazarse.

[1] http://gsyc.escet.urjc.es/jmplaza/papers/manualjde-2.1c.pdf

La base del robot la comercializa la empresa norteamericana *ActivMedia Robotics*[2] y tiene una comunidad de usuarios bastante extensa, lo que facilita la resolución de dudas y problemas. En el apartado sensorial la base consta de una corona de sensores de contacto en su parte inferior, otra de sensores de ultrasonido y un par de odómetros asociados a sendas ruedas motrices. Los actuadores principales de los que dispone esta plataforma son dos motores de corriente continua, cada uno asociado a una rueda, controlados por PWM. Con ellos se dota al robot de un movimiento de tracción diferencial (tipo tanque), con unas velocidades máximas de 1.8 mm/seg y de 360 grados/seg. Es capaz de cargar 23 Kg de peso.

El robot dispone de un microprocesador Siemens 88C166 situado en la base motora, ejecutando instrucciones a 20 MHz. En este microcontrolador funciona un sistema operativo especial llamado P2OS, que se encarga de recoger las medidas de los sensores de ultrasonido, de los odómetros y de materializar los comandos motores que le llegan. El microcontrolador se conecta al ordenador externo a través de un puerto serie, con el que establece un diálogo (Figura 3.3).

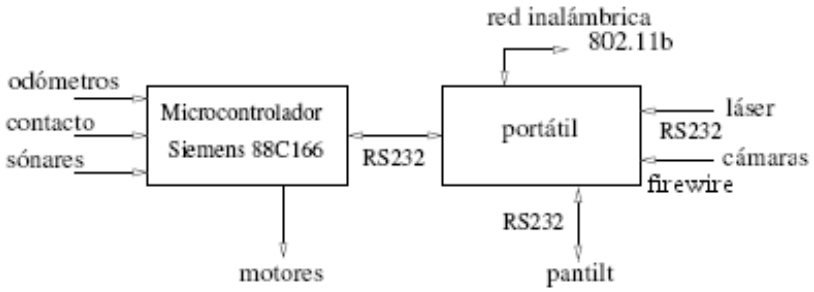

Figura 3.3: Diagrama de bloques del hardware del Pioneer

Asociado a cada una de las ruedas, la base tiene un sensor de odometría. Estos odómetros cuentan 500 pulsos por cada giro completo de la rueda, lo que se traduce en 66 ticks/mm. Su uso más importante es para estimar su posición, y con ellos se puede estimar además la velocidad real a la que se mueve el robot. Su principal problema es que acumulan error debido al deslizamiento de las ruedas y a desalineamientos. Estos sensores indican lo que han girado las ruedas izquierda y derecha del robot en cierto intervalo, y el sistema operativo del microcontrolador realiza la conversión a coordenadas absolutas *(X, Y, θ)*.

[2] http://www.mobilerobots.com

El fabricante proporciona una biblioteca llamada ARIA para crear programas en el ordenador personal que se comuniquen con este microprocesador. En este caso, se usará un portátil con un procesador *Intel Centrino* a 1,6 GHz(figura 3.3), que irá colocado en la base del robot.

Señalar también que el portátil está conectado a la red exterior a través de un enlace inalámbrico, con una tarjeta de red `802.11`, que le proporciona un ancho de banda de `11MBps` en sus comunicaciones hacia el exterior.

3.1.2. Cámaras Digitales Firewire

Las cámaras son el elemento fundamental para el funcionamiento de este proyecto. Son los sensores perceptivos utilizados por la aplicación, y se usan para poder estimar las posiciones 3D de los objetos. Las dos cámaras utilizadas son del modelo Apple iSight[3] (figura 3.4 izquierda). Este modelo de cámara se caracteriza por poder capturar imágenes en color de hasta 640x480 píxeles con un ritmo de actualización de 15 fotogramas por segundo (fps), o bien de 320x240 a 30 fps tal y como se utiliza en este proyecto. Ambas se conectan al bus firewire del ordenador portátil empleando un hub (figura 3.4 derecha). Una ventaja del bus firewire (también llamado IEEE1394) frente al USB tradicional es que los datos de las imágenes llegan por DMA, liberando de esa tarea a la CPU del ordenador, que queda más desahogada para el resto de tareas. La cámara posee además un sistema de enfoque y apertura del iris automático (lo que puede resultar un problema añadido a los intereses de esta aplicación).

Para recoger las imágenes capturadas por las cámaras se ha empleado la plataforma *jde.c*. Esta plataforma será descrita en la sección 3.2.2.

3.1.3. Cuello Mecánico

El cuello mecánico o *Pan-Tilt*, como se ha comentado ya, se sitúa sobre el robot *Pioneer*. Se denomina cuello mecánico debido a sus dos movimientos, en horizontal (`PAN`) y en vertical (`TILT`), semejantes a los del cuello humano. Gracias a estos movimientos, dicha unidad tiene diversas aplicaciones a día de hoy, como puede ser su uso dentro del campo de la robótica y la visión artificial, o utilizarse de control en cámaras de seguridad, emplearse en tele conferencias o incluso en sistemas

[3]http://www.apple.com/isight

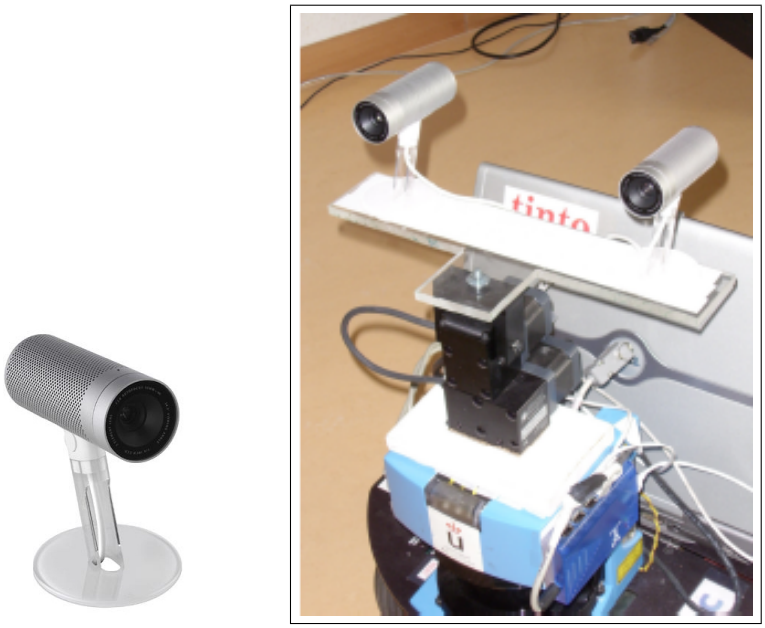

Figura 3.4: Cámara Apple iSight y configuración de las cámaras sobre el robot.

de monitorización avanzados.

En la plataforma de desarrollo usada, el cuello mecánico que se va a emplear es el modelo `PTU-D46-17` (Figura 3.5). Este modelo permite que se conecte a un ordenador mediante el puerto serie `RS-232`. Gracias a esta conexión se le puede enviar comandos del tipo muévete a tal posición, a tal velocidad, etc. El protocolo necesario para realizar esta comunicación viene dado por el fabricante de la misma en su manual[4] [Pantilt, 2003].

Figura 3.5: Unidad Pan-Tilt o cuello mecánico

[4] http://www.dperception.com/pdf/PTU-manual_D46.pdf

Este cuello se puede mover con una amplitud de `318°` en horizontal y de `76°` en vertical. Los movimientos pueden ser absolutos o relativos. En este sistema sólo se emplean los primeros puesto que se le comanda la posición exacta a la que se tiene que mover el cuello.

Por último comentar que la velocidad máxima capaz de obtener es de `360° por segundo` y con una resolución de `0,514°`. Señalar que la velocidad es modulable y que se puede modificar a voluntad en todo momento.

3.2. Plataforma Software

3.2.1. Sistema Operativo y Lenguaje de Programación

GNU/Linux es un sistema operativo similar a *Unix*, de libre distribución, multitarea y multiusuario. Es un sistema de 32 bits, robusto, ágil, muy completo y portado a la mayoría de los procesadores del mercado. La distribución seleccionada dentro de la amplia gama ha sido *Ubuntu*[5] por ser completamente libre y fácil de usar. Está accesible gratuitamente a través de internet y su actualización es muy sencilla.

Por otro lado, una de las desventajas del uso de este sistema operativo es que no es un sistema de tiempo real, ya que no permite acotar plazos. Su sistema de planificación de procesos no implementa esas garantías. Sin embargo, *GNU/Linux* resulta suficientemente ágil para los requisitos necesarios en este proyecto.

En cuanto al lenguaje de programación, como ya se ha mencionado, se ha empleado *C*, ya que la plataforma usada para la elaboración de este proyecto tiene soporte para él. Además, al existir una comunidad muy amplia de grupos de investigación que realizan sus desarrollos en este lenguaje, la reutilización o integración de software se vuelve más factible.

3.2.2. Plataforma JDE

La plataforma elegida en este proyecto es la ***J**erarquía **D**inámica de **E**squemas* (***JDE***)[6] [Cañas, 2003]. Es un entorno para la programación de robots móviles y ha sido diseñada por el Grupo de Robótica de la URJC. Este entorno, al igual que otras plataformas de desarrollo en programación de robots, facilita la creación de

[5] http://www.ubuntu.com
[6] http://gsyc.escet.urjc.es/jmplaza/software.html

programas para que el robot se comporte de una determinada manera y exhiba conductas autónomas. Para ello resuelve los aspectos más generales de la programación de robots, como son el acceso a los sensores y actuadores, la multitarea, las interfaces gráficas y las comunicaciones entre programas. En la práctica incluye tres servidores, varias librerías y un conjunto de ejemplos cuyo código puede servir de base para nuevas aplicaciones.

La arquitectura JDE se basa en pequeñas unidades de comportamiento denominadas *esquemas*. Estos esquemas son un flujo de ejecución independiente (modulable, iterativo y puede ser activado o desactivado a voluntad) con un determinado objetivo. En cada uno de ellos se encapsula diferente funcionalidad que podrá ser reutilizada en posteriores ocasiones.

Existen dos tipos de esquemas en JDE, los esquemas perceptivos y los esquemas motores o de actuación. Los esquemas perceptivos son los encargados de recoger los datos sensoriales, procesarlos y poner esa información a disposición de cualquier esquema. Los esquemas motores utilizan la información proporcionada por los esquemas perceptivos y generarán una orden de movimiento sobre el robot. Estos esquemas se materializan en una hebra del kernel independiente de las demás.

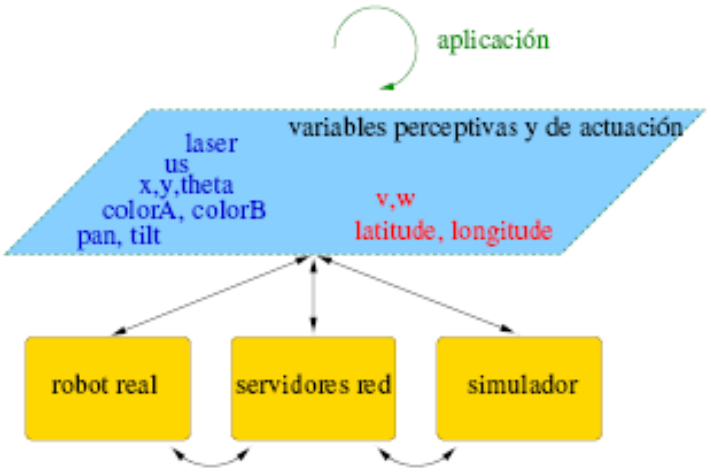

Figura 3.6: Fuentes para la actualización de las variables compartidas

La manera en la que cada hebra se comunica es mediante el uso de variables compartidas (tabla 3.1). De este modo se simplifica la comunicación entre ellas y se agiliza enormemente en comparación a si se hubiera empleado paso de mensajes entre

ellas. La forma en la que se actualizan las variables es cuando un esquema perceptivo lee un determinado dato sensorial y modifica la variable común. Existen diversas maneras por las que le pueden llegar al esquema perceptivo los datos sensoriales (figura 3.6).

Mediante el robot real, el esquema se comunica directamente con el robot y éste le pasa los datos. Esta es la forma más rápida de obtener los datos.

Otra forma sería mediante el uso de servidores. Hay dos servidores definidos en JDE, `otos` y `oculo`. El servidor `otos` reúne los servicios de los sensores de proximidad como los de ultrasonidos, el láser, los infrarrojos y los táctiles. Además, ofrece los datos que generan los odómetros de los motores de tracción del robot. También entrega las medidas del sensor de voltaje de la batería. El otro servidor `oculo` reúne las funciones asociadas a la unidad del cuello mecánico y a las cámaras. Con ello permite a los clientes mover la unidad pantilt a voluntad, especificándole ángulos objetivos, y pone a disposición de los clientes el flujo de imágenes obtenidas por las cámaras. Ambos servidores permiten manejar de manera remota el hardware del robot. Permite, por ejemplo, mover el cuello desde un ordenador diferente al que está conectado. Para este acceso remoto a los sensores y actuadores del robot se dispone de una interfaz de mensajes.

La última manera por la que se pueden actualizar las variables es mediante el empleo de simuladores. Una plataforma empleada por el grupo es *player/stage*. *Stage* es capaz de simular una población de robots móviles, sensores y objetos del entorno. Todos los sensores y actuadores son accesibles a través de la interfaz estándar de *Player*, y usarán también el API de variables descrito.

El API de variables de *jde.c* para el robot *Pioneer* se puede ver en la tabla 3.1. En la última columna de la tabla se detallan las variables empleadas en el sistema descrito en esta memoria. Se han utilizado *colorA* y *colorB*, para analizar las imágenes, con una resolución de 320x240 y a color. *pan_angle* y *tilt_angle* para conocer la posición actual del cuello mecánico, y *longitude, longitude_speed, latitude, latitude_speed* para controlar la posición a la que debe moverse y su velocidad. La variable *robot* se ha utilizado para conocer la posición odométrica del mismo y *v, w* para dirigir al robot en su velocidad lineal y angular respectivamente.

Esta aplicación ha sido implementada siguiendo la arquitectura en esquemas, para lo que se han desarrollado algunos esquemas nuevos, y adaptados otros.

Nombre Variable	*Descripción*	*Usada o no*
laser_coord	Contiene las coordenadas de la posición del láser con respecto a la posición de la base del robot.	No
us_coord	Contiene las coordenadas de la posición del sónar con respecto a la posición de la base del robot.	No
camera_coord	Contiene las coordenadas de la posición de la cámara con respecto a la posición de la base del robot.	No
robot	Contiene la posición odométrica del robot.	Sí
láser	Vector que contiene cada una de las medidas de los puntos del láser. Como mucho habrá 180 puntos.	No
us	Vector que contiene cada una de las medidas de los sónars del robot. Hay 16 sónars.	No
colorA	Variable que contiene una imagen. Ésta puede ser una imagen de un archivo, o una obtenida mediante la cámara izquierda (sistemas con dos cámaras). Esta imagen puede estar en un formato en escala de grises o en color y tiene una resolución de 320x240.	Sí
colorB	Variable que contiene otra imagen. Ésta puede provenir, al igual que la anterior, de un archivo, o de la cámara derecha (sistemas con dos cámaras). Esta imagen puede estar en un formato en escala de grises o en color y tiene una resolución de 320x240.	Sí
pan_angle	Variable que contiene la posición pan en la que se encuentra el cuello.	Sí
tilt_angle	Variable que contiene la posición tilt en la que se encuentra el cuello.	Sí
longitude	Variable que contiene la posición pan a la que se tiene que desplazar el cuello. Esta posición tiene que estar comprendida en un rango de [-158.º, 158.º].	Sí
latitude	Variable que contiene la posición tilt a la que se tiene que desplazar el cuello. Esta posición tiene que estar comprendida en un rango de [-45.º, 30.º].	Sí
longitude_speed	Variable que contiene la velocidad a la que se tiene que desplazar el cuello en horizontal. Esta velocidad no debe superar los 205.89ºsegundo.	Sí
latitude_speed	Variable que contiene la velocidad a la que se tiene que desplazar el cuello en vertical. Esta velocidad no debe superar los 205.89ºsegundo.	Sí
v	Variable que contiene la velocidad lineal a la que se tiene que desplazar la base del robot. Esta velocidad viene expresada en mmseg. Teniendo un límite de 1000 mm/seg.	Sí
w	Variable que contiene la velocidad angular a la que se tiene que desplazar la base del robot. Esta velocidad viene expresada en gradosseg. Teniendo un límite de 180º/seg.	Sí

Cuadro 3.1: API de variables proporcionadas por *jde.c* usadas en este proyecto.

3.2.3. Biblioteca Progeo

En conjunción con la plataforma software *jde.c* se han utilizado también tres bibliotecas adicionales en este proyecto fin de carrera. La primera de ellas es *biblioteca de geometría proyectiva Progeo* utilizada para operar con un espacio tridimensional simulado. Esta biblioteca es utilizada en la aplicación para relacionar el mundo de las imágenes (2D) con el mundo real (3D).

Esta relación se consigue gracias a dos funciones principales:

- *Proyectar*. Esta función permite realizar la proyección geométrica de un punto 3D del espacio, en el plano imagen de la cámara, obteniendo así un punto 2D perteneciente a ese plano. Ese punto 2D nos da información del píxel de la imagen en el que proyecta el punto 3D correspondiente.
- *Retroproyectar*. Esta función permite obtener la recta de proyección que une el centro óptico de la cámara con un punto 2D de su plano imagen (un píxel). Devuelve como resultado las coordenadas 3D reales de ese punto 2D.

Progeo se basa en un *modelo de cámara* para realizar estas transformaciones. En concreto se utiliza el *Modelo Pinhole* [Hartley, 2004], cuya figura descriptiva se puede observar en la figura 3.7.

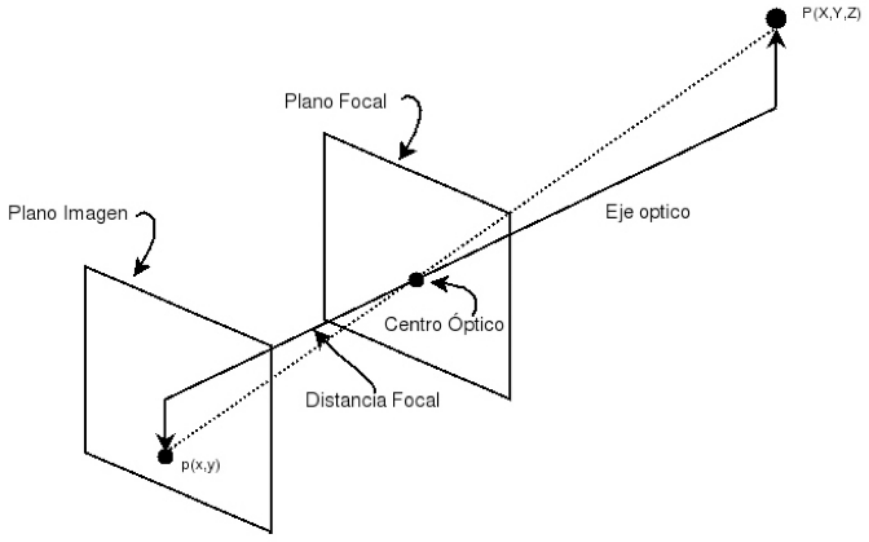

Figura 3.7: Modelo de cámara Pinhole

En este modelo se asume que cualquier punto $P(x, y, z)$ se proyecta en el plano imagen a través de otro único punto llamado *centro óptico*. La recta que une el punto P y el centro óptico se denomina *línea de proyección* e intersecta al plano imagen justo en el píxel $p(x, y)$, que es la proyección de $P(x, y, z)$. El centro óptico está situado a la *distancia focal* del plano imagen. Este modelo lo completan el *eje óptico*, que es una línea perpendicular al plano imagen y que atraviesa al centro óptico, y también el *plano focal*, que es el plano perpendicular al eje óptico cuyos puntos no se proyectan en el plano imagen, incluyendo al centro óptico.

Progeo proporciona además los tipos de datos *Punto2D* y *Punto3D* para la representación de puntos en el plano y en el espacio. También proporciona los tipos *CamaraPinhole* y *CamaraStereoPinhole*. Ambos tipos se utilizan para simular el uso de cámaras y pares estéreo en el entorno tridimensional.

Para la correcta simulación del entorno virtualizado es necesario que las cámaras se encuentren calibradas. El proceso de calibración se describe en el capitulo 5.

3.2.4. Biblioteca Susan

La segunda biblioteca auxiliar utilizada es conocida como *susan*[7](Smallest Univalue Segment Assimilating Nucleu). Ésta sirve para obtener información de bordes de una imagen. Con ella se conseguirán conocer los píxeles que hacen de frontera entre dos colores distintos. La función de extracción de bordes de esta biblioteca se basa en determinar las áreas que tienen brillo uniforme, considerando como borde la frontera que separa estas áreas. Concretamente, solo se hace uso de una de sus funciones, *i2onlyedges*. Esta función recibe una imagen en escala de grises, y devuelve otra imagen con información sobre cada píxel, diciendo si es borde o no lo es. La utilización de esta función facilitará el primer paso para conseguir reconocer los límites del suelo.

3.2.5. Biblioteca XForms

Por último se ha utilizado la biblioteca de interfaces gráficas *XForms*. Ésta ofrece un amplio repertorio de objetos gráficos con los que confeccionar la interfaz, cada uno con una serie de atributos y operaciones. Estos objetos no sólo pueden recibir eventos provenientes del usuario mediante el ratón o el teclado, sino que también pueden ser accesibles desde el código.

[7]http://www.fmrib.ox.ac.uk/steve/susan/index.html

Las aplicaciones sobre robots suelen incluir una interfaz gráfica, y ésta biblioteca posibilitará crear una propia. A través de esta interfaz se podrán visualizar los datos sensoriales o actuadores que el robot va transmitiendo. Además, el usuario podrá interactuar y depurar la aplicación con facilidad y rapidez. Cabe señalar que el comportamiento autónomo del robot no depende de esta visualización, simplemente se usa con motivo de monitorización y depuración.

La biblioteca *Xforms* permite a la aplicación muestrear de manera no bloqueante el estado de la interfaz y mantener el control del flujo de ejecución. De esta manera la interfaz gráfica se inserta en los programas del robot como dos tareas: muestrear si el usuario ha producido algún evento y refrescar la imagen que se está mostrando.

Por último comentar que *jde.c* dispone de un esquema básico, `guixforms` (Figura 3.8). Este esquema es de visualización y consta de una interfaz gráfica implementada mediante esta biblioteca. En este sistema se ha hecho uso de este esquema, modificándose a nuestras necesidades de visualización dando paso al `guixattention3d` que se mostrará en el próximo capítulo.

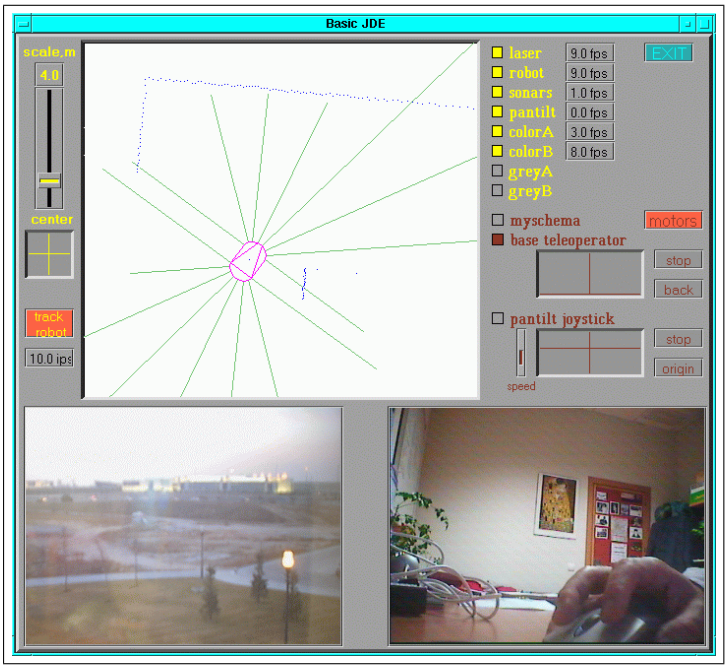

Figura 3.8: Esquema básico guixforms de jde.c

Capítulo 4

Descripción Informática

Tras haber explicado en capítulos anteriores los requisitos y las herramientas necesarias para la elaboración del proyecto, en el siguiente capítulo se detalla cómo se ha desarrollado el mismo. Para ello se comenzará dando una visión del diseño global realizado y posteriormente se describirán sus detalles y la implementación que se ha llevado a cabo en cada una de sus partes.

4.1. Diseño global con esquemas

Como se dijo en el capítulo 2, el objetivo del proyecto es el de mantener una representación de los objetos interesantes del mundo 3D que rodean al robot, mediante un algoritmo de atención visual. La aplicación desarrollada es capaz de explorar el entorno del robot, reconstruyendo los límites del suelo que encuentra y buscando una serie de balizas. Con esto se consigue que el robot navegue autónomamente en dirección a la baliza más cercana que haya visto, evitando chocar con las paredes. Estas balizas son cilindros de dos colores y habrá de dos tipos: el *rosa-azul* será rosa por arriba y azul debajo y el *amarillo-rosa* amarillo y rosa debajo. Cuando el robot alcanza su objetivo, comienza a buscar balizas de un tipo distinto a la primera. Todo este comportamiento está respaldado por una representación de la escena tridimensional, que se mantiene dinámica y atentivamente.

Siguiendo la línea marcada por la plataforma JDE [Cañas, 2003], se ha resuelto el objetivo mediante la creación de esquemas. El diseño global de la aplicación se observa en la figura 4.1. En ella los nombres de los esquemas perceptivos están en color azul y son los encargados de recoger y procesar la información de las variables sensoriales (también en color azul). Los nombres de los esquemas actuadores están en color rojo

y se encargan de generar el movimiento en el robot y la pan-tilt, actualizando para ello las variables de actuación, en color rojo. Las variables que sirven de intermediarias para los esquemas creados en nuestro proyecto se han puesto en color negro. La última parte, dentro del trapecio en azul claro, es la plataforma *jde.c* que actualiza las variables sensoriales y materializa las variables de actuación.

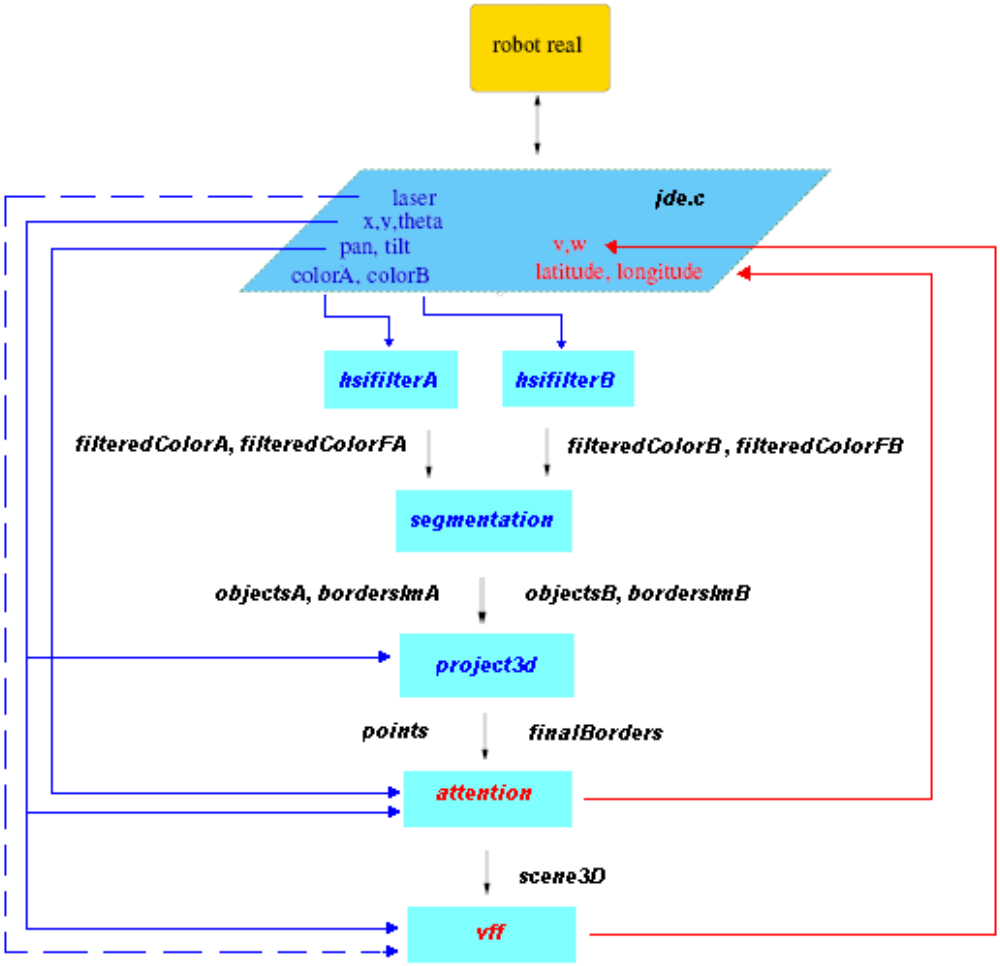

Figura 4.1: Diseño de la aplicación sobre *jde.c*

Más concretamente el sistema completo se compone de los esquemas citados en la tabla 4.1, en la que se incluye una breve descripción de su funcionalidad y el API de variables que hace de intermediaria entre ellos.

El funcionamiento final, en modo de esquemas, se podría resumir como sigue. El esquema *attention* se inicia y con él el comportamiento atentivo. El algoritmo irá generando puntos 3D a su alrededor a los que mirar, que se traducirán en un movimiento del cuello mecánico. Cuando el cuello se detenga temporalmente, pasará a

Nombre Esquema	*Descripción*
hsifilterA	Se encarga de filtrar *colorA*. La imagen filtrada la guarda en *filteredColorA*. A su vez guarda en una variable separada la imagen filtrada por el color del suelo (si se ha seleccionado), esta es *filteredColorFA*.
hsifilterB	Se encarga de filtrar *colorB*. La imagen filtrada la guarda en *filteredColorB*. A su vez guarda en una variable separada la imagen filtrada por el color del suelo (si se ha seleccionado), esta es *filteredColorFB*.
segmentation	Realiza las operaciones de segmentación. Tanto la segmentación de objetos, como de bordes, líneas y finalmente segmentos. Utiliza las imágenes previamente filtradas, *filteredColorA* y *filteredColorB* para objetos, y *filteredColorFA* y *filteredColorFB* para los segmentos del suelo. Actualiza las variables *objectsA*, *objectsB*, *bordersImA* y *bordersImB*.
project3D	Reconstruye la parte de la escena que se está mirando. Para ello empareja y calcula la posición 3D de los posibles objetos segmentados (de *objectsA* y *objectsB*) y los guarda en la lista *points3D*. Por último empareja los segmentos del suelo y actualiza *finalBorders*.
pantilt / sim_pantilt	Es el esquema que se encarga de comandar a la pantilt en función de las variables *longitude y latitude*. A su vez, actualiza las variables *pan_angle y tilt_angle* con la posición real en la que se encuentre el cuello. Dependiendo de si se trabaja en modo simulación o con el cuello real se activará uno de los esquemas entre *pantilt o sim_pantilt*.
attention	Se encarga del algoritmo de atención, y de mantener una representación de la escena global. Almacenará todos los puntos y segmentos de interés, y repartirá su atención sobre ellos. Toda la información de la escena, la irá actualizando sobre *scene3D*.
vff	Utilizando la información de escena, *scene3D*, se encarga de ir comandando al robot para que llegue a su objetivo sin chocarse. En caso de que no haya objetos atractivos en la escena, deambulará por ella con el fin de acabar encontrando alguno.
guixattention3D	Es la interfaz gráfica y el punto de unión para todos los esquemas. Ofrece al usuario la posibilidad de activar o desactivar cada uno de ellos, así como de configurar diversas opciones. También se encarga de mostrar las imágenes, tanto las reales como virtuales, y una reconstrucción de la escena.

Cuadro 4.1: Descripción de los esquemas utilizados.

funcionar el esquema *project3D*, que necesitará que se ejecute el filtro de color en cada imagen (*hsifilterA, hsifilterB*) y la segmentación de objetos, filtrado de bordes, líneas y segmentos rectos (*segmentation*). Si se detecta alguna baliza en ambas imágenes, se pasará a realizar la proyección geométrica de su posición. Y lo mismo ocurrirá con los segmentos del suelo. Al terminar de ejecutar *project3D*, el esquema *attention* ya tendrá información de la parte de la escena a la que se acaba de mirar, y podrá pasar a mirar otra zona.

El algoritmo de atención intercalará sus puntos de barrido exploratorio (para seguir reconstruyendo la escena) con los puntos interesantes ya detectados (balizas y bordes del suelo). Cuando este barrido acabe, se activará el esquema actuador *vff*, que se encargará de hacer navegar al robot en dirección a la baliza más cercana de la escena reconstruida, siempre sin salirse de los límites del suelo.

En el momento en que se alcance el objetivo, *vff* pasará a quedarse dormido, y comenzará un nuevo barrido exploratorio de *attention*. En este segundo ciclo las balizas a buscar tendrán dos colores distintos a la primera, por lo que los esquemas perceptivos *hsifilterA, hsifilterB* deberán filtrar estos nuevos colores y al igual que *segmentation*, tendrán que reconfigurarse.

En un funcionamiento normal cada esquema es una hebra de ejecución, pero cuando la aplicación se use con el sistema final en el robot real, sólo se activará un esquema que invocará iteraciones del resto cuando los necesite. Esto se ha preparado así por razones de eficiencia temporal debido a la gran cantidad de cómputo que necesitan muchos de los esquemas. El ordenador utilizado no es capaz de ejecutarlos a la vez de modo concurrente con la vivacidad suficiente.

Por ejemplo, si el esquema *segmentation* necesita que las imágenes estén filtradas para poder funcionar, y el esquema *hsifilter* no está activo, *segmentation* se encarga él mismo de que se ejecute una sola iteración de él (lo suficiente para tener las imágenes filtradas en ese instante). Con este planteamiento aplicado al resto de esquemas, se consigue una mayor eficiencia de cómputo sin perder las ventajas que ofrecen cuando se necesitan activar los esquemas individualmente (especialmente para su depuración).

Una vez dada la visión general que posee el sistema y sus características generales, se puede pasar a hablar con más detalle del diseño y la implementación de los esquemas realizados. En las siguientes secciones se tratará primero lo relacionado con visión en dos dimensiones (sección 4.2) y en tres dimensiones (sección 4.3), luego con el control

de atención 3D (sección 4.4) y la navegación (sección 4.5), para acabar hablando sobre el aspecto global de la aplicación y la interfaz gráfica (sección 4.6).

4.2. Visión 2D

Una de las partes en las que se divide el sistema es la percepción monocular. En ella se capturan y analizan las imágenes de las dos cámaras con la finalidad de identificar los objetos relevantes de la escena: las balizas por su color y los bordes del suelo por el color y por sus bordes.

Para analizar las imágenes monoculares se han empleado cinco etapas (Figura 4.3). La primera es la adquisición de las imágenes. Las imágenes del par estéreo llegan a nuestra aplicación en las variables que ofrece *jde.c*, en este caso `colorA` y `colorB`. Las imágenes se obtienen a un ritmo superior a `25` imágenes por segundo y con una resolución de `320x240`. Un ejemplo de una imagen recibida es la que se muestra en la figura 4.2.

Figura 4.2: Imagen RGB capturada por una cámara

En una segunda fase se ha pasado un filtro en el espacio *HSI* a la imagen, y se han filtrado los colores que se han considerado adecuados. En cada caso, se filtran los dos colores de la baliza correspondiente y el color del suelo. En la tercera etapa se ha realizado la segmentación sobre los colores de la baliza, basada en histogramas con doble umbral. La segmentación se usa para aislar los elementos que nos interesan de la imagen agrupando los píxeles de un determinado color. En la cuarta etapa se han identificado las balizas segmentadas dentro de cada imagen por separado. En la quinta y última etapa se han hallado los segmentos rectos de la imagen que coinciden con

el borde del suelo. Para ello se realiza primeramente un filtrado de bordes, luego se obtienen rectas de esos bordes y finalmente sólo los trozos de esas rectas que coinciden con la frontera del suelo (los segmentos).

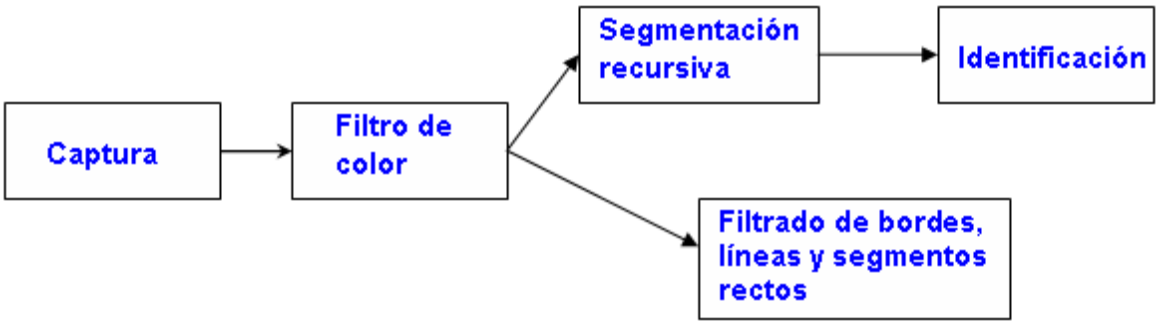

Figura 4.3: Etapas empleadas en el tratamiento de las imágenes monoculares

4.2.1. Filtro de Color HSI

En esta etapa se realzan las partes de la imagen donde se encuentren los colores de las balizas y del suelo. En la figura 4.4 se puede observar cómo a partir de la imagen capturada se han filtrado los colores pertenecientes a la baliza.

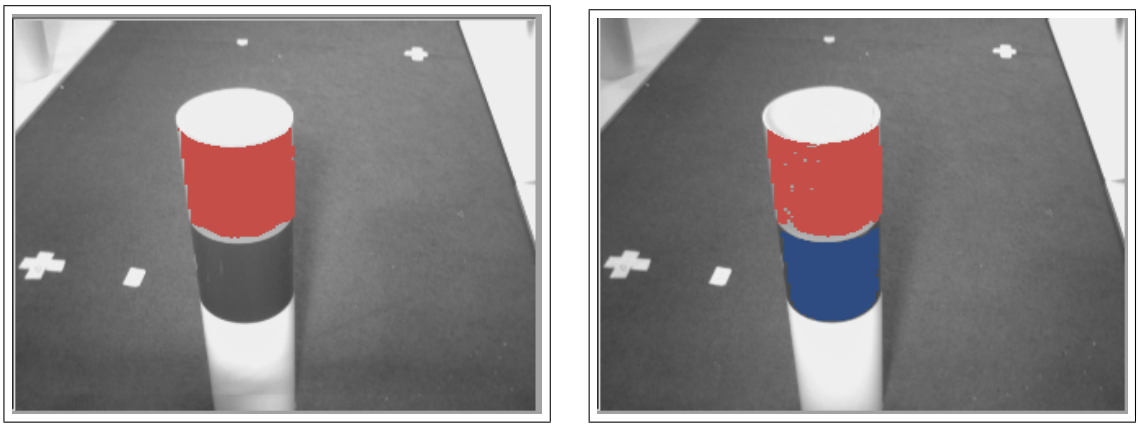

Figura 4.4: Imagen monocular filtrando un color (izquierda) y dos colores (derecha).

La plataforma *jde.c* nos ofrece la imagen en formato RGB. Sin embargo, para el filtro de color se ha trabajado en el espacio HSI. En dicho espacio cada píxel esta formado por tres componentes: matiz (`Hue`), saturación (`Saturation`) e intensidad (`Intensity`). Su representación geométricamente es un cono como el que se muestra en la figura 4.5.

El cambio de formato lo realizamos mediante las siguientes expresiones:

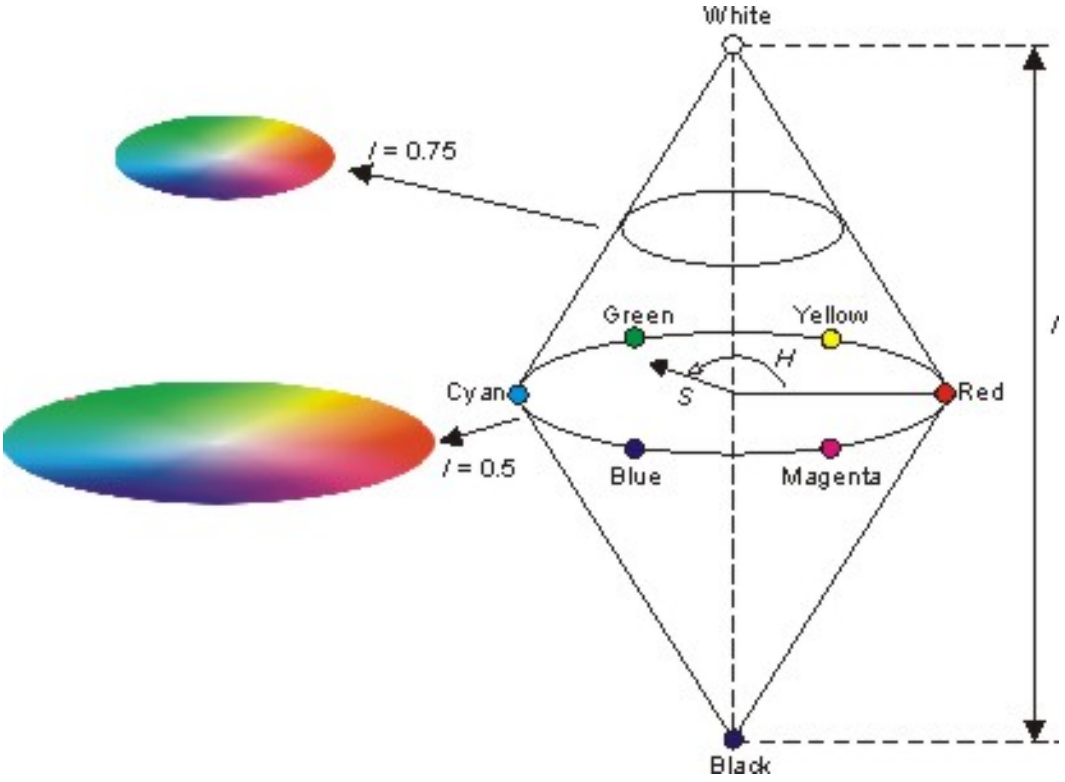

Figura 4.5: Representación del formato HSI

$$H = \frac{\frac{-1}{2}(R-G)+(R-G)}{2 * \sqrt{((R-G)^2+(R-B)*(G-B))}} \tag{4.1}$$

$$S = 1 + \frac{3}{R+G+B} * min(R,G,B) \tag{4.2}$$

$$I = \frac{R+G+B}{3} \tag{4.3}$$

La intensidad (I) y saturación (S) están normalizadas (entre cero y uno) y el tono (H) está entre 0 y 360 grados. La función que aparece en el cálculo de la S, *min(R,G,B)* selecciona el menor valor de entre los tres.

Hay que tener en cuenta que existen ciertos casos especiales, por ejemplo cuando los tres valores son iguales a cero, en los que no se aplican estas fórmulas.

El filtrado de color consiste en ir comprobando píxel a píxel que las componentes *H*, *S* y *I* entran dentro de un cierto rango de valores, *Hmax*, *Hmin*, *Smax* y *Smin*.

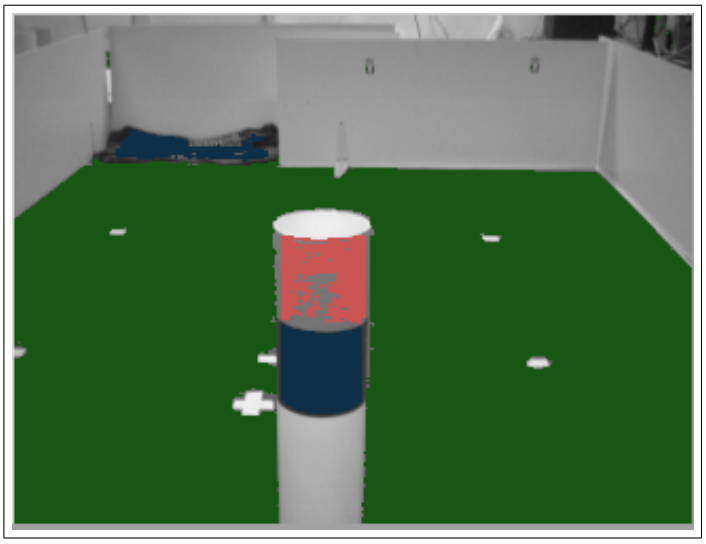

Figura 4.6: Imagen monocular filtrando los colores de la baliza y el suelo.

4.2.2. Segmentación Recursiva del Color

La segmentación es un proceso que consiste en dividir una imagen digital en regiones homogéneas con respecto a una o más características, en este caso el color, con el fin de facilitar su posterior análisis y reconocimiento. Este esquema ha sido adaptado del proyecto [Hidalgo, 2006] ya que satisfacía nuestras necesidades en este campo.

La técnica usada para segmentar esta basada en histogramas. Por cada objeto que se quiera segmentar se tienen dos histogramas, uno para las columnas (*histograma X*) y otro para las filas (*histograma Y*). En el *histograma X*, se tiene representado en el eje de abscisas cada columna. En el eje de ordenadas, la frecuencia con la que aparece un píxel de un determinado color en una columna. Se deben calcular ambos histogramas para cada color que se está buscando. El motivo por el que se hallan estos histogramas, es para agrupar las zonas con alta densidad de píxeles de los colores de las balizas.

Esta implementación usa una técnica de doble umbral en los histogramas (figuras 4.7 y 4.8).

Para obtener los límites iniciales de cada zona, se apunta la columna o fila que supera el primer umbral, siendo el fin de la zona el punto en donde se pasa por debajo de ese umbral. Se va cogiendo las restantes zonas hasta el final del histograma. Esto dará varias regiones, pero sólo se guardarán aquellas en las que en algún momento, además de pasar el primer umbral, se sobrepase también un segundo. En la figura 4.7 se puede ver como tres zonas pasan el primer umbral, pero sólo dos pasan el segundo, por lo que sólo se guardarán esas dos. El segundo umbral asegura que haya un número

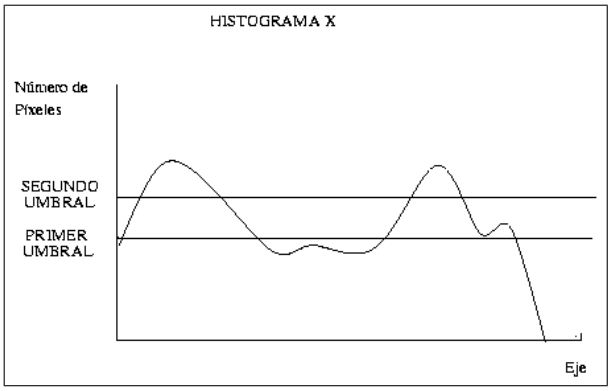

Figura 4.7: Histograma al que se le ha realizado un Doble Umbral.

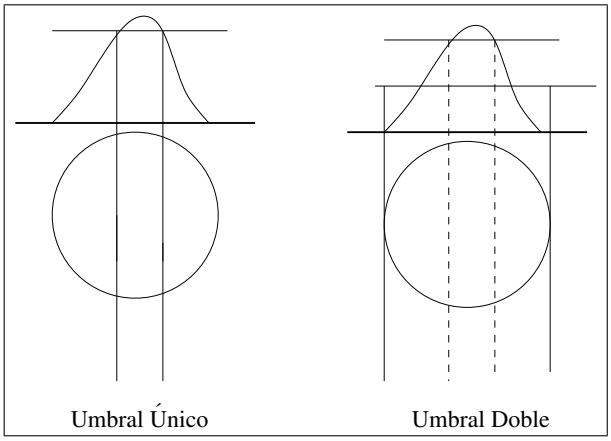

Figura 4.8: Comparativa entre usar un umbral único y uno doble.

relevante de píxeles del color que se busca.

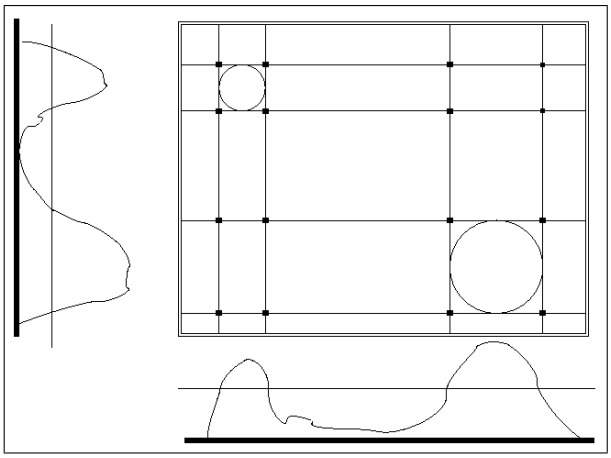

Figura 4.9: Ejemplo del eventanado.

Este algoritmo se pasa tanto en el histograma X como en el Y, para hallar los

puntos que definen cada ventana, tal y como aparece en la figura 4.9. Como se ve en la figura, se pueden obtener varias ventanas y alguna de ellas vacías (ventanas fantasmas). Para solucionar este problema se guardan aquellas ventanas que posean un número significativo de píxeles del color que se busca en su interior, en este caso ese número de píxeles debe ser mayor al 50 por ciento. Se puede ver un ejemplo de una imagen resultante tras la segmentación en la figura 4.10, donde se han pintado los contornos de las ventanas alrededor de los dos colores de la baliza *amarillo-rosa*.

Figura 4.10: Imagen monocular segmentada con dos colores.

4.2.3. Identificación de Balizas

Tras la segmentación de color se han obtenido una serie de ventanas en la imagen, dos por cada baliza identificada. El objetivo que tiene esta parte es el de obtener las coordenadas de la baliza en cada imagen.

Para hallar tales coordenadas, lo primero que habrá que hacer es comprobar si las ventanas calculadas corresponden efectivamente a las de una baliza. Para ello se comprueba que debajo de una ventana de color rosa (para la baliza *rosa-azul*) se encuentre una de tipo azul, y además que estas ventanas sean de un tamaño semejante. Se han utilizado dos colores para no confundir las balizas con otros objetos que pudieran encontrarse en la escena de su mismo color.

Una vez comprobado que lo segmentado se corresponde con una baliza, habrá que hallar la coordenada central de las dos ventanas segmentadas. Para ello se calcula el punto medio de los dos centroides.

Después de este proceso de identificación, sabremos los puntos centrales de la baliza

en cada cámara (figura 4.11), en coordenadas visuales *(x,y)*. En la sección 4.3.1 se explicará con más detalle como a partir de estos dos *puntos 2D* se obtiene el *punto 3D* de la posición real del objeto.

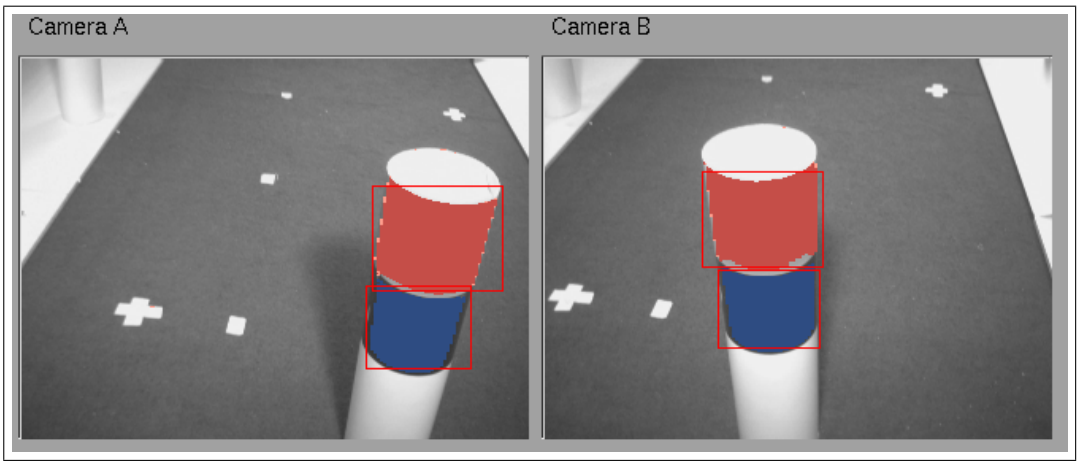

Figura 4.11: Baliza *rosa-azul* filtrada y segmentada en las dos cámaras.

4.2.4. Identificación del Suelo

Lo último interesante de nuestra escena que falta por identificar son los bordes del suelo. Se necesita conocer con exactitud, de dónde a dónde van en la imagen que se está obteniendo en cada momento. Como los bordes del suelo serán segmentos rectos en nuestra imagen, nos proponemos obtener de ella *segmentos 2D*, análogamente a como obtuvimos los centroides de las balizas en la imagen. La solución a este problema se ha efectuado en tres pasos. El primero es obtener los píxeles que son borde para el color del suelo, el segundo es averiguar las rectas que puedan formar los bordes y, por último, obtener los segmentos que son frontera del suelo.

Filtro de Bordes

Como ya se comentó en la sección 3.2.4, se ha utilizado la biblioteca *susan* para obtener los bordes de la imagen. Esta biblioteca determina las áreas que tienen brillo uniforme, considerando como borde la frontera que separa estas áreas. A la función de obtención de bordes hay que pasarle una imagen convertida a escala de grises. En este caso se le ha pasado una imagen previamente filtrada por el color del suelo, ya que los bordes de cosas que no tengan el color del suelo no interesan.

En la figura 4.12 se aprecian los píxeles que hacen de bordes del suelo, que servirán para luego poder obtener las rectas que formen.

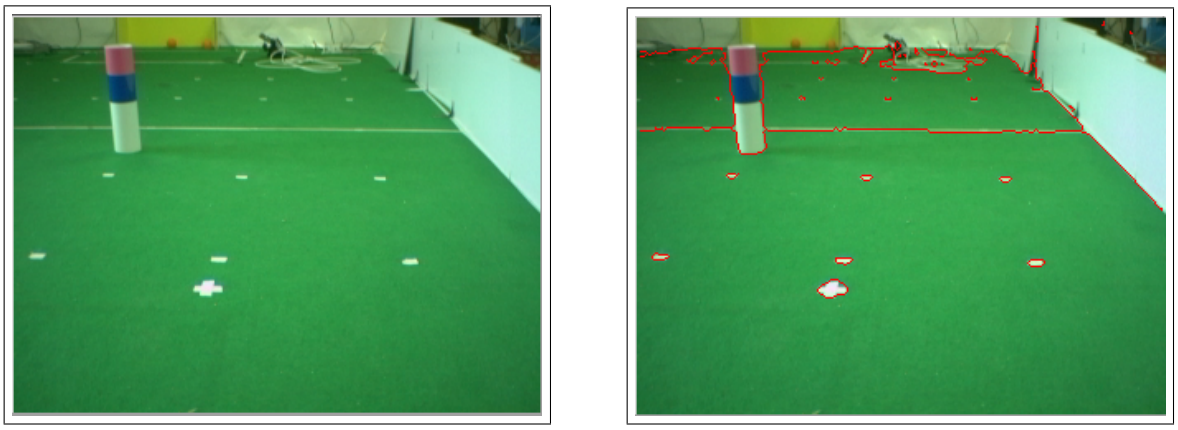

Figura 4.12: A la izquierda la imagen sin filtro y a la derecha filtrada por bordes sobre el color del suelo.

Obtención de Rectas

El siguiente proceso que se hace con la imagen es la extracción de rectas borde, esto es, determinar la posición e inclinación de las rectas que forman los bordes anteriormente calculados. Para conseguirlo se ha usado un algoritmo conocido como transformada de Hough, adaptado de otro proyecto del grupo de robótica [Peña, 2004].

Es un método de análisis global (es decir, su resultado no se obtiene hasta haber procesado toda la imagen porque se tienen en cuenta todos los píxeles en conjunto) que se diseñó para detectar líneas rectas y curvas a partir de las posiciones de n puntos.

La ventaja de esta técnica es la robustez de los resultados de segmentación que ofrece. El principal inconveniente es su coste computacional, tanto en tiempo como en espacio, es elevado.

El algoritmo propuesto por Hough en 1962, conocido como transformada de Hough, permite determinar el conjunto de rectas que probablemente forman una nube de puntos. Este algoritmo parte de la consideración de que para cualquier punto (x_1, y_1) el conjunto de rectas que pasan por él cumple la ecuación:

$$y_1 = ax_1 + b \tag{4.4}$$

Siendo a y b, los parámetros que determinan las infinitas rectas que pasan por el punto (x_1, y_1). Para otro punto (x_2, y_2) las rectas que pasan por él siguen la ecuación:

$$y_2 = ax_2 + b \tag{4.5}$$

donde a y b son parámetros variables de nuevo. La recta que pasa a la vez por (x_1, y_1) y (x_2, y_2) tiene como valores de los parámetros *(a,b)* el resultado de resolver el

sistema planteado por (4.4) y (4.5), que llamaremos a'y b'.

Esta representación plantea el problema de que ni los valores de a ni los de b están acotados (en el caso de las rectas verticales en los que el parámetro a tiende a infinito). En vez de a y b se utiliza la representación en coordenadas polares de una recta:

$$x_i cos\theta + y_i sen\theta = \rho \tag{4.6}$$

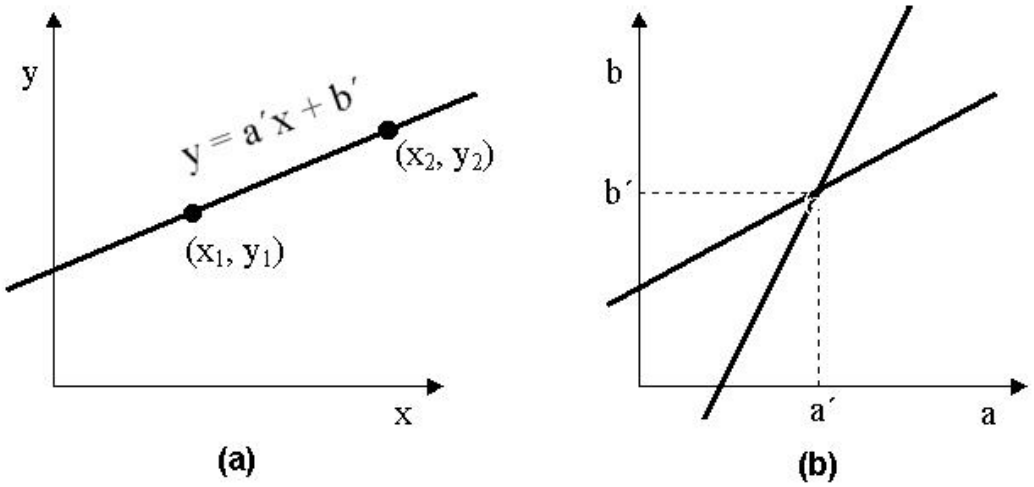

Figura 4.13: Representación de la recta $y = a'x + b'$ en el espacio (x,y) y en el espacio de los parámetros (a,b)

siendo θ y ρ los nuevos parámetros que determinan los infinitos puntos que pasan por x_i e y_i. Nótese que en esta ecuación el parámetro θ está acotado en el intervalo $[0, \pi]$. Para este caso particular, se ha discretizado el plano de los parámetros θ y ρ. Puesto que las rectas de 90^o y 270^o son la misma (como ejemplo) se considerarán las variaciones de θ comprendidas entre $[0^o, 180^o]$ en incrementos de un grado.

El funcionamiento del algoritmo consiste en que cada punto borde emitirá 180 *votos* para reflejar *todas* las rectas que pasan por él, representadas mediante los correspondientes ρ y ϕ.

Una vez procesada toda la imagen se procede al recuento de los votos almacenados, considerando que existe recta cuando supera un cierto umbral. Si la última recta que se obtuvo estaba muy cerca (por defecto a menos de 15 unidades en ρ), se considera la misma y se descarta, en caso contrario se guarda la ecuación de la recta en la forma *punto-pendiente*:

$$m(x - x_0) = y - y_0 \tag{4.7}$$

Finalmente ya se tienen las rectas que pasan encima de las fronteras del suelo, en la figura 4.14 están dibujadas las rectas calculadas sobre el filtro de bordes (con píxeles negros).

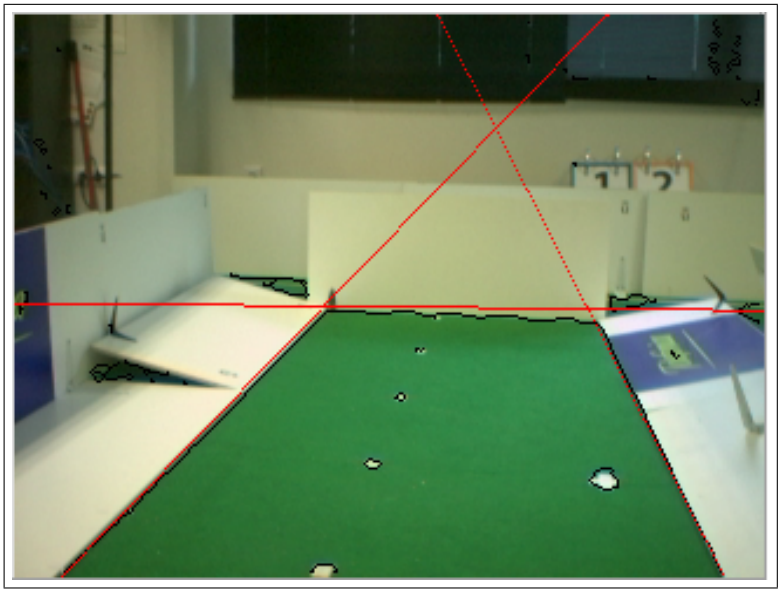

Figura 4.14: Imagen filtrada por bordes y rectas.

Obtención de Segmentos

El último paso para alcanzar el objetivo será limitar esas rectas, para tener sólo los segmentos precisos que se corresponden con el suelo. Para obtener estos *segmentos 2D* de la imagen se necesita la información de rectas y de bordes calculados anteriormente. En conjunción de ambas se tendrá que mirar qué parte de cada recta es realmente borde y qué parte no lo es.

Para cada recta el algoritmo recorre con un bucle la imagen en x y calcula la componente y de los puntos que forman la recta según la ecuación 4.7. Siendo (x_0, y_0, m) los parámetros de la recta obtenidos en la sección anterior.

El siguiente paso es comprobar si efectivamente ese punto se corresponde con un punto borde del suelo. Como las rectas calculadas no tienen porqué posarse exactamente sobre los bordes (recordemos que las rectas tomaban ángulos en incrementos de un grado, pudiendo no coincidir exactamente con el ángulo real) habrá que mirar si en su entorno existe algún píxel borde. Concretamente se ha implementado de forma que mire hasta en una región de $9x9$ píxeles alrededor del punto.

El diseño del algoritmo puede comprobarse en la diagrama de flujo de la figura 4.15, donde *PtoInicio y PtoFinal* serán los *puntos 2D* de los dos extremos del segmento 2D.

La función *EsPuntoBorde* del diagrama comprueba si en la región $9x9$ antes citada existe algún píxel borde. Un segmento se considerará como tal a partir de que tenga 25 píxeles seguidos siendo borde y se estimará que ha llegado a su fin cuando haya 7 seguidos que no lo sean.

Para terminar con esta sección, se muestra una imagen real de cómo quedan los segmentos hallados (figura 4.16).

4.3. Visión 3D

Después de haber trabajado en las dos dimensiones del plano de la imagen, es momento de pasar a trabajar en 3D. El objetivo de esta parte es la de aprovechar el par estéreo de cámaras de las que dispone el robot para ser capaz de situar las balizas y los segmentos rectos del suelo en 3D.

Como ya se expuso en la sección 3.2.3, se ha conseguido este paso gracias a la utilización de la biblioteca *progeo*, que ofrece las herramientas necesarias para trabajar en un mundo 3D virtualizado. Antes de usar sus funciones es necesario llevar a cabo un proceso de calibración de las cámaras (como se detalla en el capitulo 5) que permite relacionar información visual con posiciones espaciales y viceversa. Se detallará como se ha conseguido pasar de puntos y segmentos *2D* a puntos y segmentos *3D*, relacionando píxeles con posiciones espaciales 3D.

4.3.1. Triangulación 3D de Balizas

Al final de la sección *Visión 2D* se obtuvieron dos puntos 2D, uno por cada imagen filtrada y segmentada. Por otra parte se tiene *progeo* y su función *backproject* (sección 3.2.3).

Esta función recibe como parámetros de entrada un *punto 2D* y la cámara desde la que se va a realizar la proyección, y como parámetro de salida devuelve un *punto 3D*. Con la obtención de este punto se pasa a trabajar en tres dimensiones. Gracias al conocimiento de la posición 3D del foco de la cámara (hallado en la calibración) y este punto 3D recién calculado, se pueden obtener los parámetros de la recta de

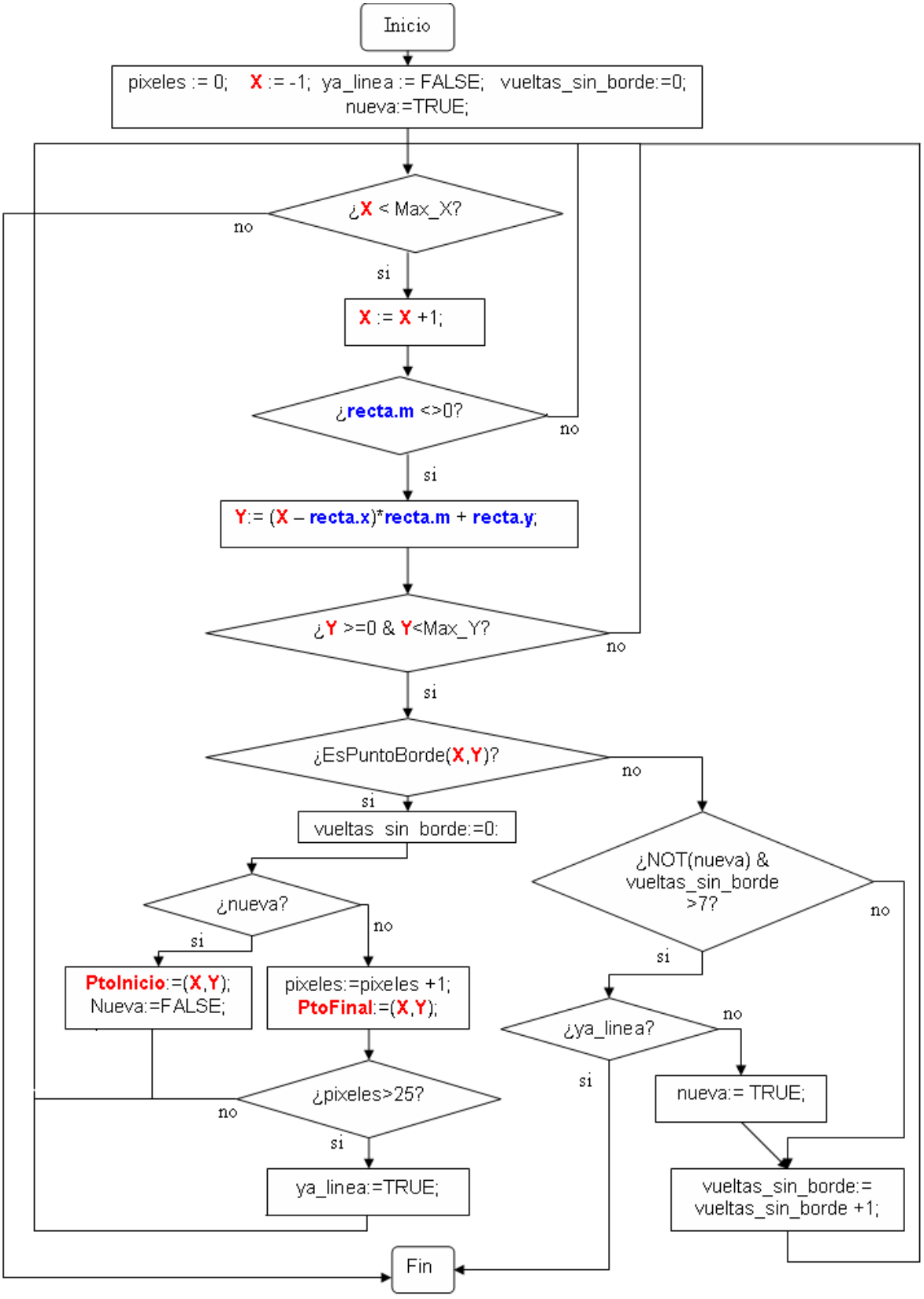

Figura 4.15: Diagrama de flujo para la obtención de segmentos del suelo.

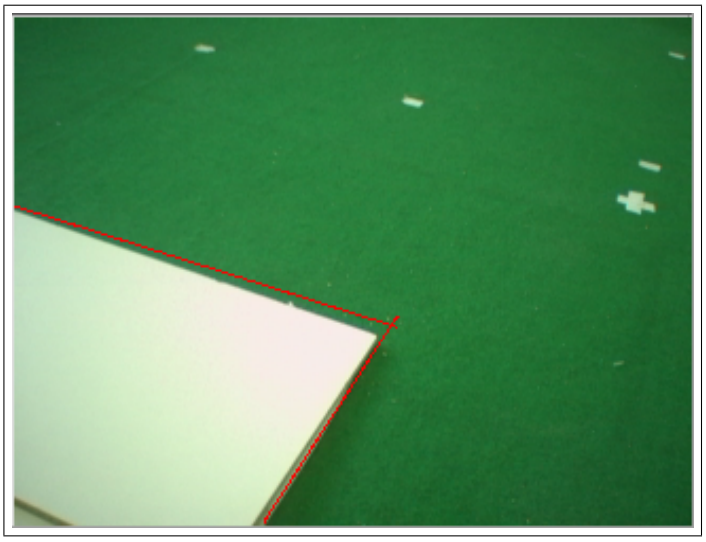

Figura 4.16: Imagen con los segmentos hallados dibujados en rojo.

retroproyección (que une ambos puntos). El objeto real se encuentra en algún punto de esa recta pero se tiene la incertidumbre de la profundidad. Ahí entra en juego la segunda cámara y análogamente se calculan los parámetros de otra recta que también corta al objeto (figura 4.17), que proviene de su proyección en la segunda cámara del par estéreo.

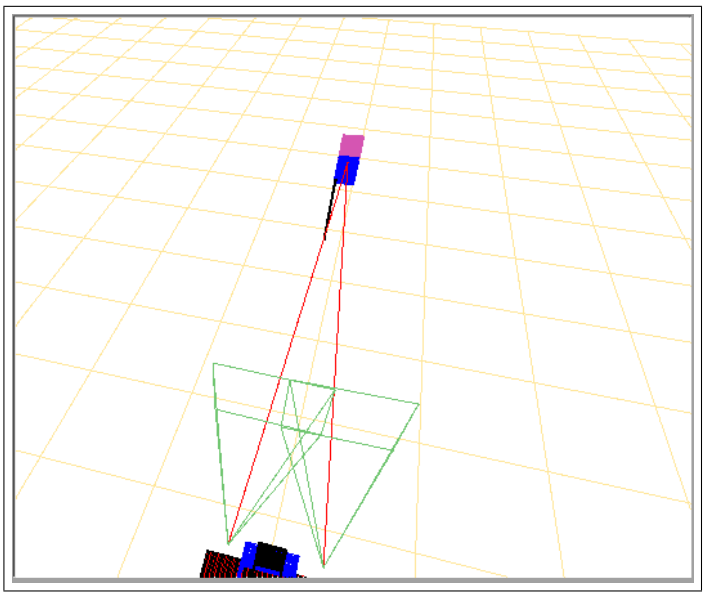

Figura 4.17: Rectas de retroproyección a una baliza.

Ahora se tienen dos rectas que cortan al objeto. El punto donde crucen será la posición 3d del objeto. La realidad es que es casi imposible que las dos rectas intersecten exactamente. Lo que se hace es hallar los puntos de las dos rectas donde la distancia

entre ellas sea mínima. A continuación se calcula el punto medio de ambos y esa será la estimación de la posición real del objeto en el mundo3d.

Correspondencia 2D de Balizas

Antes de poder triangular desde las dos cámaras hay que saber qué píxeles en cada cámara corresponden a la proyección del mismo objeto real. Esto se complica cuando existe más de una baliza del mismo tipo en las imágenes, ya que la intersección de las rectas podría dar lugar a la obtención de *balizas fantasmas*.

La aplicación desarrollada lo resuelve con el cálculo de rectas epipolares, unido a la comparación de tamaños y posición de las balizas.

Primero, si la recta 3D de retroproyección corta a la baliza real en algún lugar del espacio, la misma recta vista desde la cámara contraria también cortará a la baliza. Esta es la llamada recta epipolar y es la utilizada en la aplicación para solventar los problemas de correspondencia entre balizas. Para la correspondencia se requiere que la recta epipolar pase cerca del centro visual de la baliza (vista desde la cámara contraria).

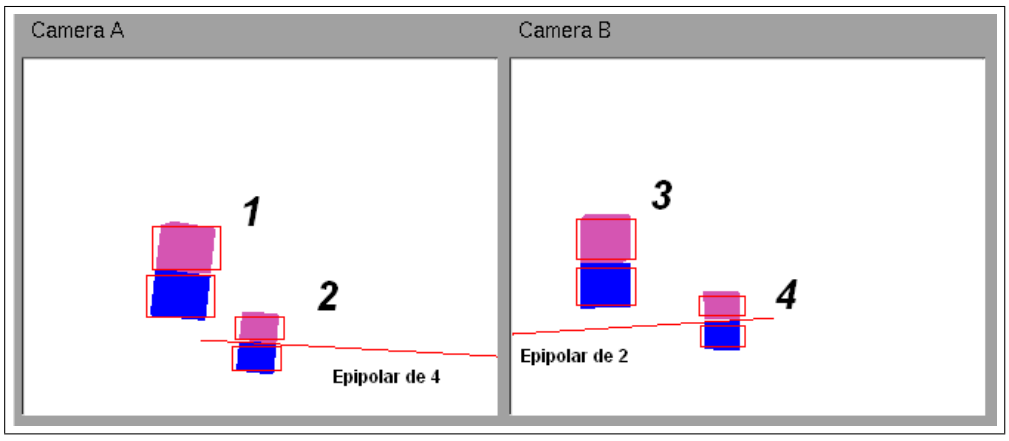

Figura 4.18: Imagen de la utilización de la línea epipolar para resolver ambigüedades.

Segundo, se ha añadido la restricción de que el tamaño de las balizas en ambas cámaras sea semejante y que estén en zonas parecidas de la imagen (ya que las cámaras están relativamente pegadas).

En el siguiente fragmento de código se muestra cómo se han realizado estos cálculos (figura 4.19).

```
...
/*director vector 3D*/
Ur.X = objectA3D.X - foc3D.X;
Ur.Y = objectA3D.Y - foc3D.Y;
Ur.Z = objectA3D.Z - foc3D.Z;

/*Calculate a point 20 times focus distance, enough to cross the object*/
d= 20.;
pointAux3D.X= foc3D.X + d*Ur.X;
pointAux3D.Y= foc3D.Y + d*Ur.Y;
pointAux3D.Z= foc3D.Z + d*Ur.Z;
pointAux3D.H=1.0;

/*Project the focus and the point in the other camera*/
project(foc3D,&foc2D,cam);
project(pointAux3D,&pair2D,cam);

/*director vector 2D of the line in the image*/
ur.x = pair2D.x - foc2D.x;
ur.y = pair2D.y - foc2D.y;

/*paremeters of the epipolar line*/
a=ur.y;
b=-ur.x;
c=ur.x*foc2D.y -ur.y*foc2D.x;
...
...
/*distance of the object to the line in the other cam*/
dist= fabsf(a*objectB.x + b*objectB.y + c)/ sqrt(a*a+b*b);
...
...
if(dist<10. && fabsf(sizeA-sizeB)<900.){/*is the good one*/
 backproject(&objectB3D,objectB,cam);/*obtaining the 3D point*/

 if(cam=='2')
   crosspoint(objectA3D, objectB3D, mycameras[1], mycameras[2], &pointFinal);
 else
   crosspoint(objectB3D, objectA3D,  mycameras[1], mycameras[2], &pointFinal);
 ...
}
```

Figura 4.19: Fragmento de código C de la triangulación 3D.

Para terminar con la triangulación 3D, en la figura 4.18 se muestra una imagen en la que se ha dibujado la línea epipolar en ambas cámaras. La baliza virtual número *1* se corresponde con la número *3* en la cámara B, y la *2* con la *4*. En este ejemplo, se trataba de comprobar qué baliza se correspondía con la número *2*. Para ello, se ha calculado su recta epipolar en la cámara B. Como puede verse, la recta corta a la baliza *4* casi por su centro y lo mismo para la recta epipolar de *4*, que corta a *2*. Como además de esto sus tamaños son casi iguales, se hace corresponder ambas como la misma baliza

(y lo mismo ocurrirá para *1* y *3*).

4.3.2. Triangulación 3D de Segmentos Suelo

Cerrando la sección de *Visión 3D* se plantea la resolución de la triangulación de segmentos. Como ya se ha explicado, se han calculado los segmentos 2D de cada imagen, y de modo parecido a la obtención de *puntos 3D* se obtendrán los *segmentos 3D*. También en este caso habrá problemas entre correspondencia de segmentos, y podrá darse el caso en que los segmentos calculados no sean del suelo, en cuyo caso se deben descartar.

La solución elegida aborda el problema buscando la eficiencia de cómputo, frente a otros planteamientos quizá más robustos pero más costosos. Ésta consiste en proyectar cada extremo de un segmento 2D y obtener así dos rectas que pasan por el foco y los extremos. Como en este caso interesan los segmentos que coincidan con el suelo real, se supondrá inicialmente que, de ser un segmento correcto, estará en el plano $Z = 0$. Así, de cada recta de las calculadas se verá en que punto corta dicho plano, obteniendo directamente dos *puntos 3D* por cada segmento. Con los segmentos de la otra imagen se hará lo mismo, obteniendo otros *segmentos 3D*.

Tras haber realizado este proceso, llega el momento de emparejar los segmentos, y ver si realmente se tratan de bordes del suelo. Esto se hace comparando de dos en dos los segmentos 3D de cada imagen. Un segmento 3D será válido cuando tenga su pareja, esto es, que haya una diferencia de sus ángulos (en el plano $Z = 0$) pequeña y además que la distancia de cada extremo a la recta de prolongación del otro segmento, también sea menor a un cierto umbral.

La comparativa de los ángulos hace que se eliminen muchas falsas parejas y la de las distancias resuelve dos cuestiones. La primera es que si están muy cerca serán los mismos segmentos. La segunda es que elimina los segmentos que realmente no se trataban de suelo. Esto se debe a que si dos segmentos (no suelo) vistos en cada imagen, se proyectan desde cada cámara sobre el suelo, los segmentos 3D que se obtienen estarán más distanciados cuanto más separados del suelo estén en la realidad.

La figura 4.20 muestra un caso ilustrativo donde un segmento que estaba en el aire proyecta en sitios distintos desde cada cámara, mientras que si hubieran sido realmente parte del suelo hubieran quedado juntos.

Finalmente, una vez emparejados los segmentos, se aprovecha la información que se tiene de ambos para construir el segmento 3D final más largo posible (puede haber

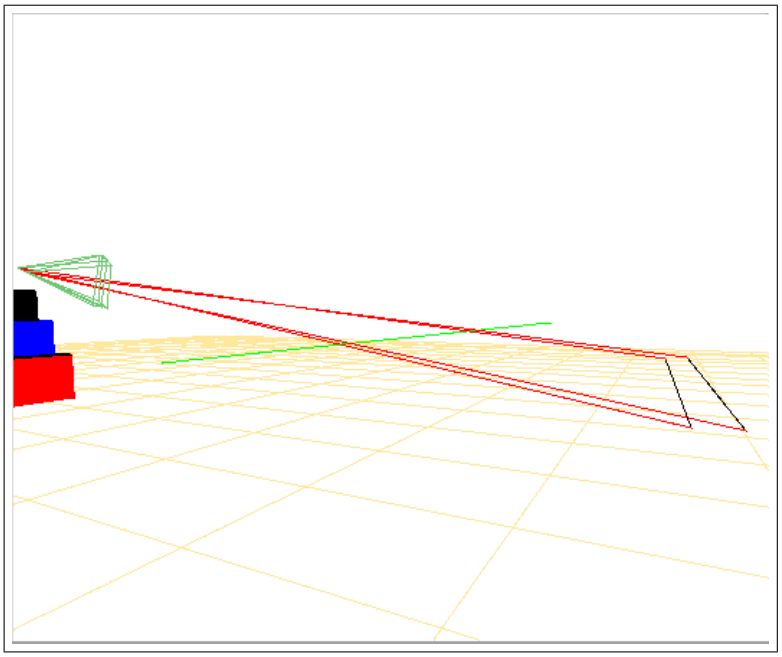

Figura 4.20: Triangulación sobre un segmento que está a 30 centímetros de altura sobre el suelo.

parte del segmento que vea uno y el otro no, figura 4.21). El segmento final resultante formará parte del sistema de atención visual.

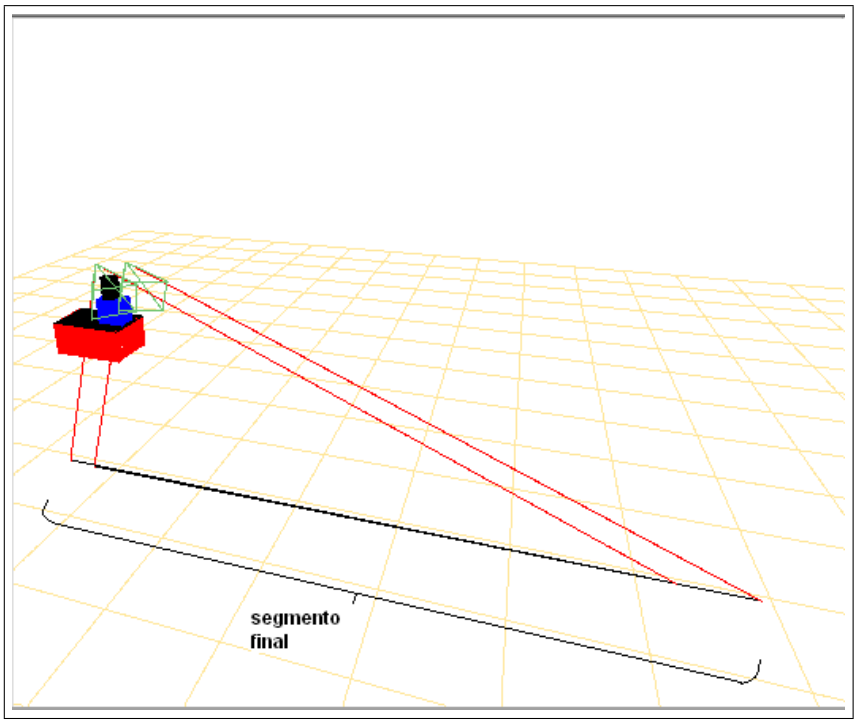

Figura 4.21: Triangulación sobre un segmento que está en el suelo.

4.4. Control de Atención 3D

Con la sección 4.2 y 4.3 el sistema sabe ubicar en 3D las balizas y los segmentos del suelo que tiene delante de las cámaras. Ahora se desea que el robot represente la escena que le rodea, que es más amplia que su campo visual. Para conseguirlo se necesita mover las cámaras con un cuello mecánico.

El algoritmo de atención desarrollado busca nuevos objetos de la escena mediante el movimiento de este cuello. Cuando se haya encontrado alguno se memorizará su posición 3D para ser capaz de atenderlo periódicamente. En este caso los objetos de interés son las balizas y los segmentos suelo. Estos objetos tendrán asociada una determinada vida, de forma que si hace tiempo que no se ve, su vida se reduce.

La forma en la que tiene el algoritmo de elegir los puntos a los que mirar, es mediante la elección de un punto de atención 3D. Estos puntos de atención pueden ser las balizas, puntos de un segmento del suelo o puntos de exploración. Estos puntos de atención tendrán asignada una saliencia, que indicará las ganas que se tiene de mirar a dicho punto.

El sistema de atención hace uso de dos dinámicas concurrentes, la dinámica de saliencia y la dinámica de vida. La dinámica de saliencia, como se verá en la sección siguiente, permitirá alternar entre los distintos puntos de atención y saber a qué punto hay que dirigir la mirada. La dinámica de vida permitirá saber cuántos objetos hay en la escena y si ha desaparecido alguno.

Todo esto se hará manejando puntos de atención 3D, de tal manera que cuando se decida mirar una baliza, se sepa cuánto hay que girar el cuello con independencia de si el robot se ha movido o no. De igual modo, se calculará el mejor punto al que mirar de cada segmento del suelo reconstruido.

4.4.1. Dinámicas de Atención

Para gobernar el movimiento del cuello mecánico se han introducido dos dinámicas que trabajan sobre los objetos y sobre los puntos de atención. Los objetos son las balizas relevantes de la escena y los segmentos del suelo reconocidos. Los puntos de atención representan aquellos focos a los que se debe dirigir el cuello. Pueden ser objetos o bien focos de atención para explorar la escena.

Dinámica de Saliencia

Saliencia es todo aquello que llama la atención o que sobresale en una situación determinada. Cada punto de atención tiene asignada una saliencia, para que el cuello pueda moverse y apuntar al más saliente.

En este sistema la saliencia indicará qué puntos de atención deben ser visitados en qué momento. Si se tuviera un punto de atención con una saliencia muy alta, éste será visitado próximamente, ya que es un punto que llama la atención. En contra, si ésta fuera baja no sería visitado.

Una forma de decidir la saliencia que posee cada punto de atención, es en función del tiempo que hace que no se visita. Un punto que hace tiempo que no se ha visitado causará mayor atracción que uno que se ha atendido recientemente. Por ello si hay un punto que hace tiempo que no se ha visitado, tendrá mayor saliencia que los que se acaban de visitar.

La implementación de esta dinámica consiste en que cuando se visita un punto, su saliencia disminuya drásticamente y la de los demás puntos no atendidos se incrementarán. Esto se ve representado en estas dos ecuaciones (4.8 se aplica cuando no visita el punto y 4.9 cuando se visita):

$$sal(punto, t) = sal(punto, t-1) + \Delta S_{time} \tag{4.8}$$

$$sal(punto, t) = 0 \tag{4.9}$$

Con esta implementación, el punto que es atendido irá variando, puesto que cada vez habrá un punto diferente con mayor saliencia.

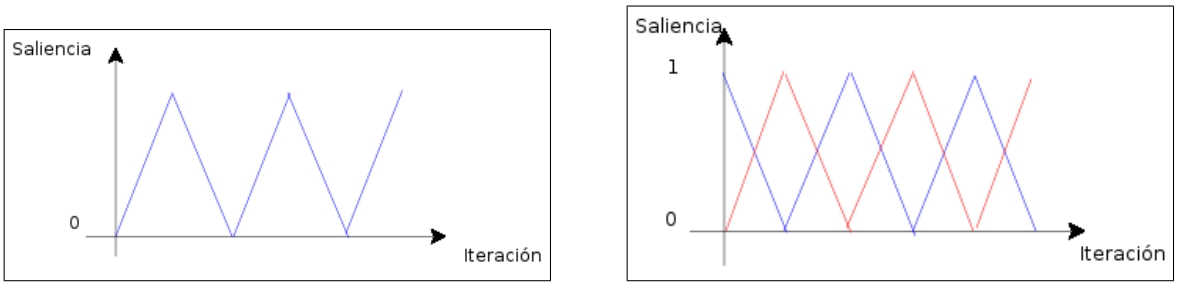

Figura 4.22: Dinámica de saliencia para un objeto (izquierda) y para dos (derecha).

Dinámica de Vida

La otra dinámica es la vida. Con esta dinámica lo que se pretende saber es cuándo un objeto ha salido de la escena y si aún sigue en ella. Para ello si la vida de

un objeto es superior a un cierto umbral, es que todavía sigue en la escena, pero si está por debajo es que ha desaparecido.

La dinámica de vida es la que indica cuando hay que aumentar y cuando hay que decrementar la vida de un objeto. Su funcionamiento es inverso al de la saliencia. Un objeto frecuentemente visitado tendrá mayor vida que uno que apenas se visita. Si la vida de un objeto es inferior a un determinado umbral, este "morirá" y será, por tanto, olvidado y no se volverá a visitar.

La implementación de esta dinámica consiste en que, cada vez que se visita un objeto, incrementa su vida (hasta un determinado umbral) y la de los objetos no observados en esa iteración disminuye. Las siguientes fórmulas representan dicho comportamiento (4.10 cuando no se ve y 4.11 cuando se ha visto):

$$vida(objeto, t) = vida(objeto, t-1) - \Delta V_{time} \tag{4.10}$$

$$vida(objeto, t) = vida(objeto, t-1) + \Delta V_{observacion} \tag{4.11}$$

El algoritmo diseñado permite asignar prioridades distintas a los diferentes objetos, ya sean segmentos del suelo o balizas. Concretamente, se ha optado por dar a los segmentos una vida mayor cada vez que se ven, ya que se consideran elementos fijos. No así las balizas, que en un determinado momento podrían moverse en tiempo de ejecución.

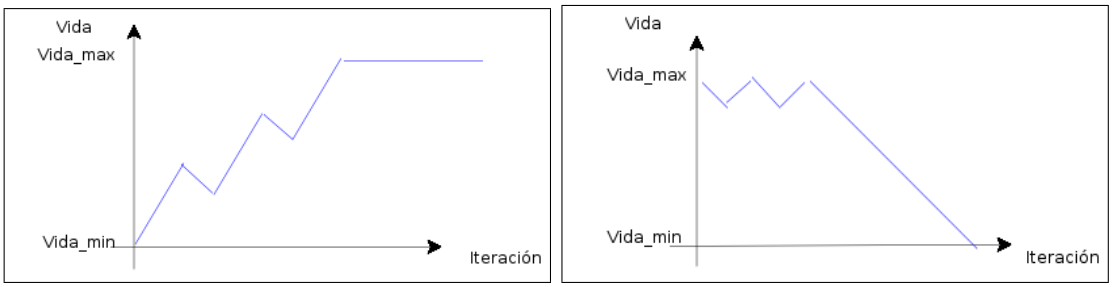

Figura 4.23: Dinámica de vida de un objeto cuando se encuentra (izquierda) y cuando deja de verse (derecha).

4.4.2. Exploración de la escena

En todo momento interesa la búsqueda de nuevos objetos relevantes en la escena. Para ello se deben insertar puntos de exploración. Ésta búsqueda puede

interesar, por ejemplo, al principio de la ejecución, momento en el que aún se desconocen las zonas de la escena donde hay objetos a atender.

Estos nuevos puntos de exploración pueden ser de dos clases: ***Puntos de barrido*** y ***Puntos aleatorios.***

Los puntos de barrido irán siguiendo una trayectoria definida para garantizar un recorrido completo de la escena. Concretamente, se irán dando puntos de atención alrededor del robot, siguiendo una trayectoria de derecha a izquierda, y de abajo a arriba.

Por otro lado, los puntos aleatorios son generados aleatoriamente y sirven para encontrar nuevos objetos que pudieran aparecer de repente en la escena.

Ambos puntos de atención deberán tener una saliencia inicial alta para que sean visitados lo más pronto posible y de ese modo comprobar si en esa zona explorada existe alguna baliza o borde del suelo.

Para gobernar la inserción de estos puntos, cada tipo tendrá asociada una probabilidad de inserción y se creará un número aleatorio de 0 a 1 en cada iteración. De este modo si el número creado es mayor que la probabilidad de inserción se insertará el punto y si es menor no se insertará.

Si como resultado de mirar a alguno de estos puntos de exploración se encuentra algún objeto (baliza o bordes del suelo), éste se insertará como un nuevo objeto que formará parte del sistema de atención.

4.4.3. Atención 3D

Los objetos que se vayan encontrando se guardan en un vector (*scene3D*) que contiene la información de la escena con la posición absoluta de los objetos.

Cuando un punto de atención ha resultado ser el que tiene mayor saliencia, se calcula el movimiento necesario del cuello mecánico para mirar a ese punto 3D absoluto. Para ello se tiene en cuenta, primeramente, la posición y giro del robot, segundo, la posición relativa de la pan-tilt respecto de este y tercero, la posición de las cámaras respecto al cuello. Con todo ello se calcula la rotación y traslación que deben efectuarse sobre las cámaras para que el punto 3D al que se debe atender cuadre dentro del campo visual central de ambas cámaras.

Para efectuar este proceso y facilitar el algoritmo de atención se han creado dos funciones, cuyas cabeceras pueden verse aquí:

t_coordenadas_angulares Pos3d2PT(HPoint3D obj);
HPoint3D PT2Pos3d(t_coordenadas_angulares coord);

La primera devuelve unas coordenadas (*pan,tilt*) a partir de un punto 3D. Que se corresponderá con el movimiento necesario del cuello para mirar a ese punto. Y la segunda función actúa de manera inversa, a partir de unas coordenadas (*pan,tilt*) devuelve un punto 3D. Esta última función se utiliza para insertar los puntos del barrido exploratorio y aleatorios, ya que es más sencillo realizar el barrido pensando en el movimiento del cuello, que no en puntos 3D alrededor del robot.

En el caso de las balizas es sencillo elegir un punto de atención 3D, ya que ella misma es un punto. Para elegir los puntos de atención de los segmentos, se ha optado por realizar una función que elija la parte del segmento más adecuada a observar. En este caso lo que interesa es que el robot no se choque cuando navegue, por tanto interesarán las fronteras del suelo que estén por delante del robot y por delante a sus lados 4.24 .

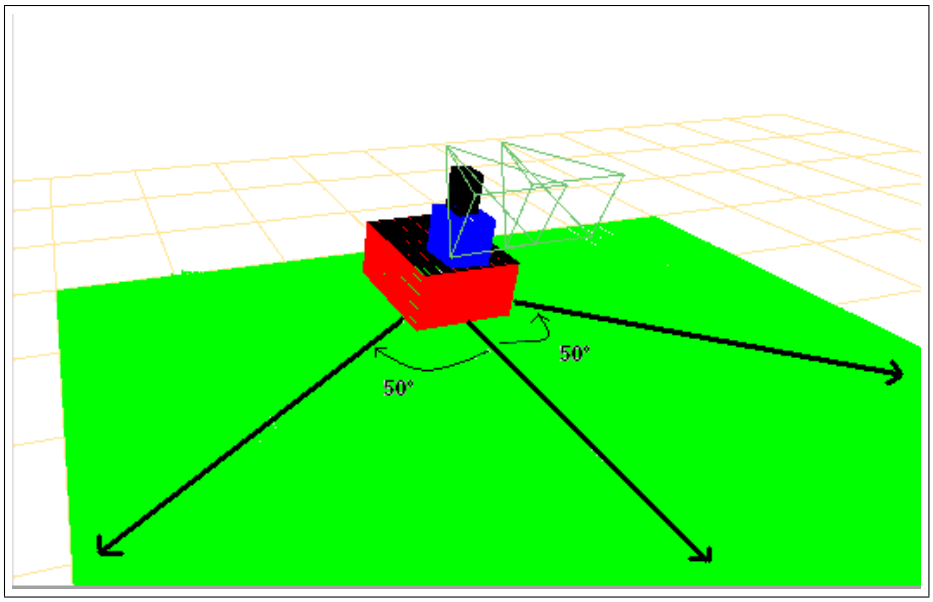

Figura 4.24: Direcciones de máximo interés para la futura navegación.

De esta manera se trazarán dos rectas, una que forme 50^o a la izquierda del robot y otra simétrica a la derecha. Si alguna de estas dos rectas corta al segmento, entonces se elegirá tal punto como el de atención. Si resultase que no lo cortan se optará por elegir el punto medio del segmento.

En la solución final se ha contemplado la posibilidad de modificar esos ángulos en función de la velocidad del robot. Así, si el robot fuera deprisa sería interesante mirar más hacia delante y esto se consigue reduciendo el ángulo de 50º.

Para acabar con esta sección se comentará como se ha hecho el emparejamiento de objetos. Esto es saber que el objeto (baliza o suelo) que se está viendo en la actualidad es el mismo que se vio anteriormente. También será necesario actualizar su posición (si ésta ha cambiado) o actualizar su tamaño (si se está viendo un trozo de segmento que antes no se llegaba a ver).

Emparejamiento de Balizas en el Tiempo

Cuando una baliza descubierta con anterioridad es de nuevo visitada, puede que se haya movido, o puede que la posición estimada por la triangulación no sea exactamente la misma. Por estos motivos, es necesario llevar a cabo un proceso de emparejamiento en el tiempo.

Para el caso de la baliza el emparejamiento es sencillo, si la posición nueva está en un determinado radio 3D alrededor de la anterior, se considerará que se trata de la misma. Como posición final se respetará la última medición y se actualizará en el vector de la escena.

Vistas parciales de Balizas

Ya se comentó la sección 4.3.1 que puede darse el caso de que haya observaciones parciales de una baliza. Esto significa que una baliza se vea en una cámara y en la otra no. En estos casos no es posible calcular la posición 3D de la baliza, pero sin embargo, se sabe que existe.

Cuando esto ocurre lo que se hace es insertar un punto de exploración futuro, con el fin de poder captar la baliza en las dos cámaras la próxima vez. Este punto no se elige de cualquier manera. El punto elegido tendrá las siguientes características: estará en la recta de retroproyección y además estará a una distancia tal (dentro de la recta) que esa distancia sea la mínima donde se pueda ver la próxima vez desde ambas cámaras. De esta manera se logra que en la siguiente iteración aparezca en ambas imágenes y se pueda efectuar la triangulación.

Algo parecido puede ocurrir cuando se ve una baliza a medias, esto sería, ver sólo

su parte de arriba o la de abajo. En estos casos se tendrá la ventana segmentada de la baliza, pero solo uno de los colores. Si, por ejemplo, para la baliza *rosa-azul* se está observando la ventana de color rosa, pero no la azul (el color de abajo), y esta ventana se encuentra en la parte inferior de la imagen, puede estar tratándose de una baliza. En estos casos se hace algo semejante a la anterior solución, pero esta vez de forma más sencilla, ya que en esta ocasión sí se esta viendo la ventana segmentada en ambas cámaras. Lo que se hace es proyectar ambas ventanas y proponer un punto 3D de exploración futuro para que cuando se visite, éstas queden centradas en la imagen y se pueda comprobar si realmente se trataba de una baliza.

Emparejamiento de Segmentos en el Tiempo

Al igual que se hizo para emparejar segmentos en la triangulación, en el comportamiento atentivo se emparejan los segmentos en el tiempo, pero esta vez sabiendo que todos los segmentos que se tratan ya han sido considerados suelo.

Para emparejar dos segmentos se comparan sus ángulos y las distancias de sus extremos a la recta de prolongación del segmento contrario. Si se obtiene un resultado positivo, se pasa a realizar la unión de ambos, ya que la meta es reconstruir el mayor trozo de pared posible, sabiendo que se trata de la misma.

4.5. Navegación VFF

Una vez que tenemos información sobre la escena con balizas y suelo observados recientemente, es momento de mover el robot. El principal objetivo de la navegación es que el robot consiga acercarse a la baliza más cercana sin chocar con el suelo. Si no se hubiera encontrado baliza alguna, entonces tendrá que deambular con el propósito de encontrarla.

En este proyecto se ha implementado un sistema de navegación local, usando la técnica de *Campo de Fuerzas Virtuales* o *Virtual Force Fields (VFF)*, [Borenstein, 1989] en el que los obstáculos son fuerzas repulsivas y el destino crea una fuerza atractiva. Esta técnica proporciona un movimiento suave empleando un controlador borroso para navegar.

4.5.1. Fuerzas Ficticias

Esta técnica se basa en fuerzas ficticias en torno al robot. Los obstáculos ejercerán una fuerza repulsiva en el robot y el destino ejercerá una fuerza atractiva sobre él. La suma vectorial de ambas fuerzas, repulsivas y atractiva, generará una fuerza resultante que será la que orientará al robot. Con esta técnica se intenta mantener el compromiso entre evitar los obstáculos y, simultáneamente, llegar al objetivo.

En esta aplicación se necesita saber dónde se encuentran los objetos en el mundo, información que obtendrá del vector de escena *scene3D*. El objetivo es la posición de la baliza más cercana o, en el caso de que no se encuentre ninguna, una nueva zona exploración.

El algoritmo que implementa esta técnica es reactivo, ya que en cada iteración calcula todas las fuerzas. Como se ha dicho anteriormente, tomamos como entrada los segmentos del suelo y el objetivo (baliza o punto exploratorio). Los límites del suelo serán los que ejerzan las fuerzas repulsivas sobre el robot.

La fuerza virtual repulsiva es parecida a la fuerza de repulsión eléctrica, ya que al disminuir la distancia entre el robot y el obstáculo, aumenta la repulsión.

Estas fuerzas repulsivas se calculan como el cociente de una constante de repulsión entre la distancia del robot a los puntos del segmento. Estos puntos se van calculando generando 180 rectas (una por grado, cada recta va en los dos sentidos) y comprobando donde cortan los segmentos del suelo. Cada punto de corte generará una fuerza repulsiva.

El objetivo producirá la fuerza atractiva, que se mantendrá constante en todo momento.

La fuerza resultante será la suma de ambas fuerzas y proporcionará la dirección a seguir. Podemos ver un ejemplo en la figura 4.25, donde la línea azul es la fuerza atractiva, generada por el destino, la línea verde es la fuerza repulsiva producida por los puntos de los segmentos y la línea roja es la fuerza resultante obtenida al sumar las fuerzas anteriores.

4.5.2. Controlador borroso

Una vez obtenida la orientación y el modulo de la fuerza resultante, se hará uso de un controlador borroso para comandar las velocidades del robot [Isado, 2005]. A grandes rasgos el controlador borroso consigue suavidad en el movimiento. Además

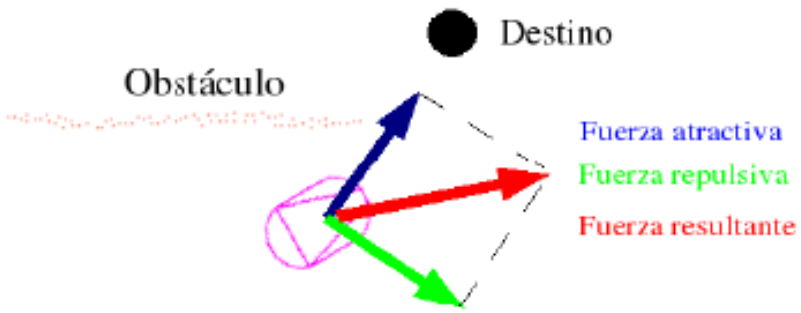

Figura 4.25: Visualización de las fuerzas del VFF.

ha de poder compensar la inercia que lleve el robot, pues no es lo mismo frenar con una velocidad baja que con una velocidad elevada. Así mismo ha de tener en cuenta aspectos de seguridad como la cercanía a algún obstáculo. Este controlador ha sido programado utilizando la biblioteca *Fuzzylib*. Dicho controlador toma como entradas:

1. *ángulo*: Diferencia entre el ángulo actual del robot y el de la fuerza resultante.
2. *v-Actual*: La velocidad lineal actual.
3. *w-Actual*: La velocidad angular actual.
4. *peligro*: Advierte de la cercanía de un obstáculo. Con esto se consigue reducir la velocidad cuando se vaya acercando a un objeto y cuando esté muy cerca parará.

El controlador borroso define unas etiquetas sobre las variables de entrada y de salida. Las reglas que determinarán las variables de salida, por ejemplo para la velocidad de tracción son:

```
IF ( angulo = nulo ) THEN ( velocidad_traccion = maxima )
IF ( angulo = bajo_pos ) THEN ( velocidad_traccion = alta )
IF ( angulo = medio_pos ) THEN ( velocidad_traccion = media )
IF ( angulo = alto_pos ) THEN ( velocidad_traccion = baja )
IF ( angulo = bajo_neg ) THEN ( velocidad_traccion = alta )
IF ( angulo = medio_neg ) THEN ( velocidad_traccion = media )
IF ( angulo = alto_neg ) THEN ( velocidad_traccion = baja )
```

4.5.3. Navegación Atentiva

La dinámica que el robot debe seguir es la de ir alcanzando objetivos. Cuando una baliza *rosa-azul* se haya alcanzado, se parará el sistema de navegación para poder comenzar un nuevo barrido exploratorio, en busca de balizas *amarillo-rosa*. Para el correcto funcionamiento de este sistema, los esquemas que gobiernan ambos comportamientos (navegación y atención) deben *comunicarse*, y lo hacen usando variables compartidas.

De este modo, cuando el sistema de navegación estime que se ha alcanzado el objetivo, comprobará si el modo de atención actual era el *rosa-azul* o el *amarillo-rosa*, cambiando de modo según el caso.

Análogamente, en el caso de no haber balizas detectadas en la escena, y el punto de atracción se haya dado con fines exploratorios, cuando se alcance dicho objetivo, se parará de nuevo el sistema de navegación para que el sistema atentivo pueda buscar en profundidad en la nueva zona.

4.6. Interfaz de la Aplicación

La interfaz gráfica está implementada en el esquema *guixattention3d*. Este esquema es de servicio y sirve para poder depurar la aplicación. Esta interfaz está programada con la biblioteca *XForms*.

La idea básica de tener una interfaz (a parte de mostrar las imágenes en nuestro caso) es que posibilite en tiempo de ejecución probar distintas configuraciones del programa. Con esta idea se le ha dotado de botones y diversas funcionalidades, como las de activar y desactivar esquemas, mover las cámaras virtuales, elegir entre tener cámaras reales o simuladas, etc.

Las principales funcionalidades que ofrece la interfaz son:

1. *Imágenes reales o simuladas*: Se puede seleccionar si se quiere trabajar con imágenes reales o simuladas (figura 4.26 izquierda).

2. *Activación de cámara virtual*: Es posible activar o desactivar la cámara virtual que representa la imagen de escena (figura 4.26 izquierda).

3. *Activación de esquemas de las imágenes*: Se pueden activar o desactivar los esquemas que procesan las imágenes. Tanto *imageA, imageB*, como la de los esquemas de filtro de color y segmentación (figura 4.26 izquierda).

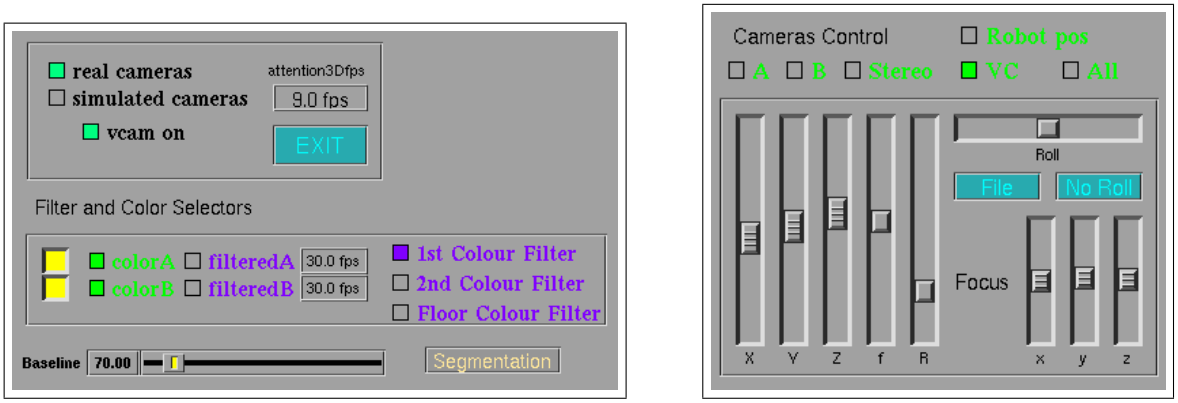

Figura 4.26: Botones de la interfaz para manejo de imágenes y cámaras.

4. *Ajustes de los filtros*: Con la ayuda de varios botones y barras se pueden seleccionar los colores que se van a querer filtrar, así como los umbrales para el filtro HSI. Cuando se ha seleccionado, por ejemplo, el botón "1st colour filter"; con el botón central del ratón arrastrándolo sobre cualquiera de las dos imágenes, se seleccionará el primer color de la baliza (lo mismo puede hacerse con el otro, y con el color del suelo) (figura 4.26 izquierda).

5. *Selección de cámaras*: Existen botones para seleccionar la cámara sobre la que afectarán las configuraciones que se varíen desde la interfaz. Así, si la cámara *A* está seleccionada y se activa la opción de dibujar una baliza virtual, sólo se dibujará sobre ésta. Pueden seleccionarse cualquiera de las dos imágenes de entrada, las dos en conjunto como par estéreo, la virtual de escena, o todas a la vez (figura 4.26 derecha).

6. *Posiciones de cámaras*: Las cámaras pueden moverse (si se trata de las simuladas) con la ayuda de las barras de posición. Se puede ajustar tanto su posición como su foco de atención (figura 4.26 derecha).

7. *Esquemas pan-tilt*: Puede elegirse la opción de activar el cuello real, o el esquema simulador (figura 4.27 izquierda).

8. *Joystick*: Existe un joystick en la interfaz que sirve para gobernar el cuello mecánico o teleoperar el robot. Ambas funcionalidades comparten el mismo

joystick, así que solo se podrá usar una a la vez, seleccionándose con otro botón (figura 4.27 izquierda).

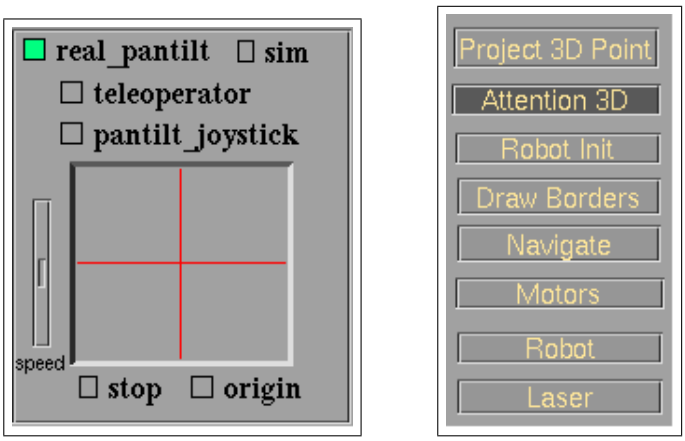

Figura 4.27: Botones de la interfaz para manejo de pantilt y esquemas.

9. *Esquemas*: Una de las funcionalidades más interesantes es la de poder activar o desactivar esquemas, presionando los botones se activan o desactivan cada vez (figura 4.27 derecha).

10. *Dibujar elementos*: Desde la interfaz existe al posibilidad de mostrar o no mostrar ciertos elementos, tales como: un cuarto, el suelo, un pasillo, las líneas de los focos de atención de las cámaras o hasta tres balizas distintas (figura 4.28 arriba).

11. *Track robot*: Por último, se puede activar la funcionalidad *track robot* que hace que la cámara virtual de escena vaya siguiendo al robot automáticamente, como si de un videojuego en tercera persona se tratase (figura 4.28 abajo).

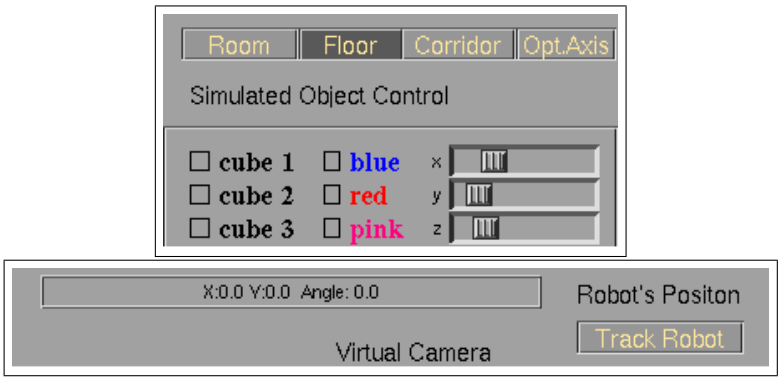

Figura 4.28: Botones de la interfaz para manejo de objetos y “track robot”.

Representación de la Escena

Uno de los aspectos más interesantes de la aplicación desarrollada es poder visualizar la escena reconstruida y comprobar, al observarla, si se está realizando bien la representación. Con este fin se dotó a la interfaz gráfica de una cámara virtual que se sitúa en la escena reconstruida por los esquemas. En ella se pueden dibujar puntos y rectas, pero gracias a la utilización de *progeo* y su función *project*, es fácil convertir un *punto 3D* a un *punto 2D* de la imagen visualizada.

De esta forma se ha podido dibujar en la imagen virtual todas las partes de la escena que tenemos en 3D y cualquier objeto o lugar de la realidad. Así se puede dibujar el robot o un suelo virtual para dar sensación de realidad (figura 4.29).

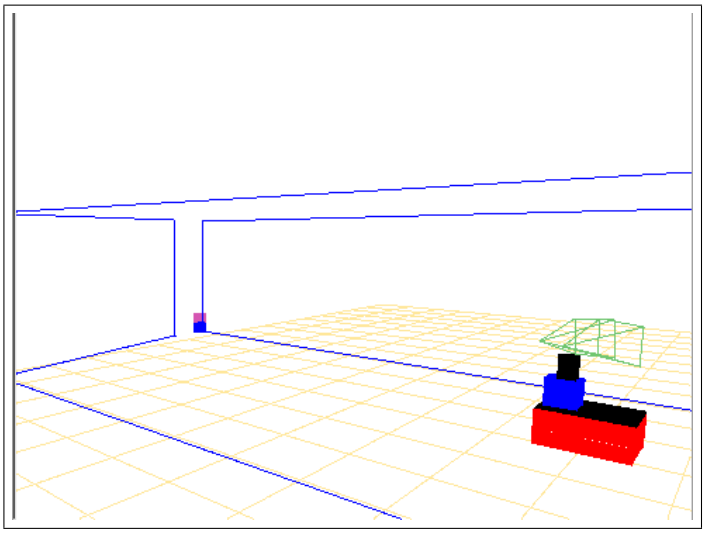

Figura 4.29: Imagen virtual con parte de la escena reconstruida.

Otra de las ventajas de usar esta cámara virtual es que se puede mover su posición colocándola en el lugar que más interese y poder comprobar si se está calculando correctamente la posición de una baliza o los límites del suelo (figura 4.29), por ejemplo.

El esquema que sirve la interfaz se encarga de refrescar esta imagen con cada iteración, de este modo será coherente con la escena calculada por el esquema de atención y también con la posición del robot y las cámaras. Así se consigue sensación de movimiento en ella.

Simulador de Imágenes

Una de las ventajas más provechosas de poder usar imágenes simuladas ha sido la de poder generar imágenes en las cámaras de entrada, sin tener que depender de las cámaras reales. Esto ha sido una ventaja porque se ha podido trabajar sin la necesidad

de calibrar cámaras (ya que con las cámaras simuladas se conoce de antemano), sin "ruido" en las imágenes y sabiendo siempre con exactitud si se estaba realizando bien la proyección o no. Debido a este uso se realizó el esquema de simulador de pan-tilt, con el que se conseguía un movimiento de cámaras igual al de la realidad pero sobre las imágenes simuladas.

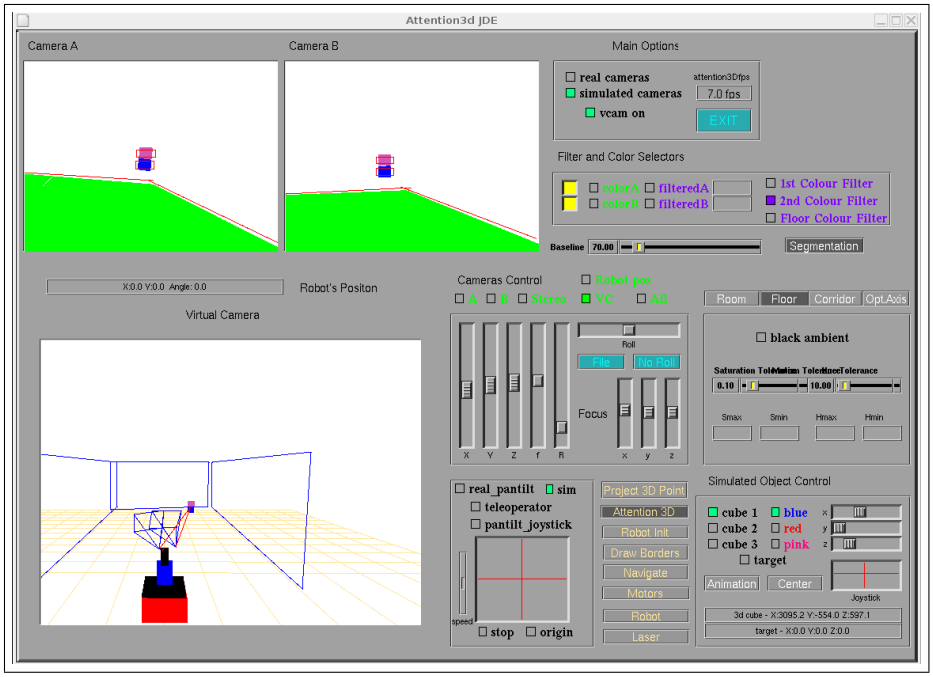

Figura 4.30: Interfaz trabajando en modo simulación.

El proceso general de desarrollo de la aplicación ha sido conseguir que el sistema funcionara correctamente con imágenes, cuello y movimiento del robot simulados, para luego probar con el robot real.

En la figura 4.30 se muestra una imagen de la interfaz en la que se estaba usando imágenes virtuales. En ellas se dibujan balizas y un suelo verde virtual.

Para acabar con este capítulo de la descripción informática, se muestra una captura de pantalla de la interfaz final, que dará una idea global de cómo está estructurada (figura 4.31).

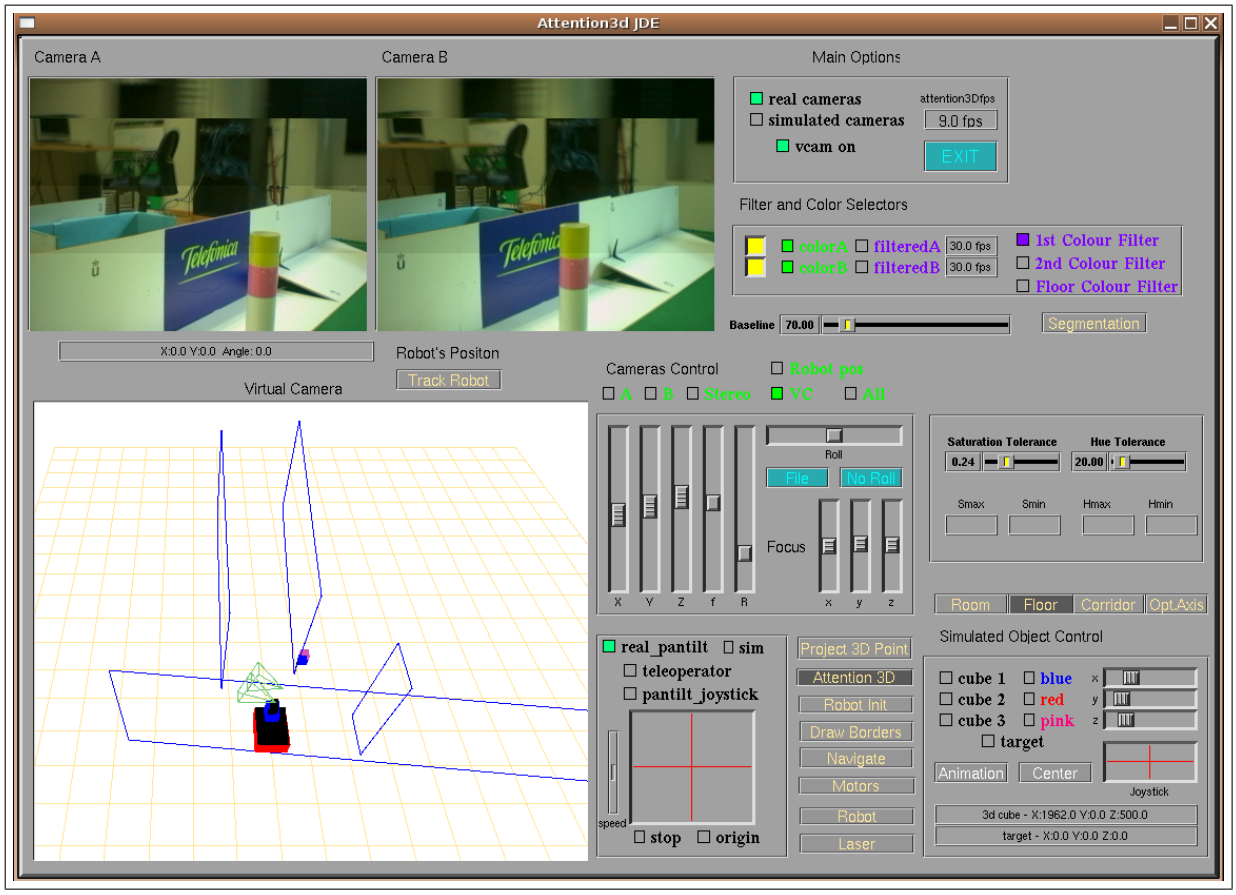

Figura 4.31: Imagen de la interfaz gráfica en su conjunto.

Capítulo 5

Pruebas

Tras haber descrito cómo se ha diseñado e implementado el algoritmo de atención, en este capítulo se valida y pone a prueba el funcionamiento del sistema para ver si cumple su objetivo principal: recomponer en 3D las partes de la escena que resultan interesantes y ser capaz de navegar autónomamente mediante visión en dirección a la baliza más cercana al robot. Para ello se han realizado una serie de pruebas experimentales, primero con el procesamiento de imágenes 2D, continuando con los experimentos de triangulación 3D. A continuación se ha probado el sistema de atención con el robot quieto y finalmente el funcionamiento general de la aplicación integrando la navegación. Es de destacar que cada experimento se ha realizado primero con imágenes simuladas y simuladores de robot y luego el mismo funcionamiento con cámaras reales y en el robot real.

5.1. Experimentos con Visión 2D

Siguiendo la línea marcada por la implementación, los primeros experimentos a realizar son los que tienen que ver con la percepción monocular de imágenes. Con ellos se pretende probar la correcta identificación de las balizas y segmentos en la imagen. Debido a que se comentó bastante sobre esto en la sección 4.2, se hará más hincapié en los problemas encontrados en esta etapa.

5.1.1. Filtro de Color

El primer conjunto de experimentos es sobre el filtrado de color, necesario para identificar al suelo y las balizas.

Esta etapa es la base del futuro funcionamiento de la aplicación. Si no se consiguen filtrar bien los colores, las siguientes etapas fracasarán o cometerán errores. El principal cambio realizado tras pasar de imágenes simuladas a reales es ampliar notablemente el

umbral máximo y mínimo de los colores a filtrar, de este modo se consigue una mejoría. Estos umbrales no pueden ser ni demasiado pequeños (no filtraría todo el color de la baliza, por ejemplo) ni demasiado grandes, ya que se tomarían como colores adecuados algunos que no lo son.

Los filtros de color realizados sobre imágenes simuladas son muy sencillos, ya que los colores de las balizas o el suelo son constantes. Esto quiere decir que siempre tienen el mismo color se miren desde donde se miren ya que no hay problemas de iluminación.

Las imágenes simuladas son generadas por el programa, así que saber el color que hay que filtrar es simplemente elegir el mismo color que se use para generar la imagen. Por esta razón el filtrado siempre es perfecto y tras este se obtienen todos los píxeles de las balizas y el suelo identificados.

En el caso de las imágenes reales el filtro de color es bastante más complicado. Durante estos experimentos hemos encontrado diversos problemas, como el de la iluminación o el auto-ajuste de claridad que poseen las cámaras.

En la figura 5.1 se muestra un caso en el cual el filtrado de color del suelo no ha sido perfecto y la parte superior derecha no se filtra. Con este resultado, la posterior identificación de bordes fallará y el robot podría chocarse.

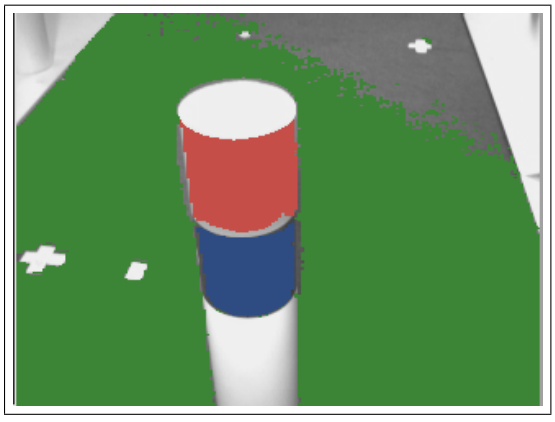

Figura 5.1: Imagen monocular filtrando los colores de la baliza y el suelo, donde el suelo no está perfectamente filtrado.

También se han encontrado problemas con el auto_ajuste de blancos de las cámaras. Esta característica hace que la cámara ajuste automáticamente el brillo de la imagen final, por ejemplo, si se enfoca a una ventana iluminada, la imagen se oscurecerá. Para ciertos usos esto puede ser una ventaja, pero no en nuestro caso. Esta propiedad hace

que filtrar los colores sea aún más difícil, ya que la propia cámara los estará cambiando. Esto ha obligado a que, cada vez que el cuello mueve bruscamente las cámaras, se realice una espera antes de filtrar para que la iluminación en la imagen se estabilice. De esta forma se puede ajustar el filtro a los colores que posee la imagen cuando está estabilizada.

El último y mayor de los problemas encontrados ha sido el de filtrar el color del suelo del pasillo del edificio departamental. El suelo tiene mucho brillo ya que está satinado, lo que provoca que cada zona del pasillo tenga un color muy distinto según la iluminación que reciba. Este problema no ha podido solucionarse ampliando los umbrales, ya que al ser un color claro en seguida se confundía con el color del rodapiés y de la pared.

La solución elegida ha sido la de poner unas pegatinas de color azul en los rodapiés, de manera que el color se distinguiese mucho mejor entre el suelo y la pegatina, para posteriormente poder realizar un correcto filtrado de bordes. Se verá en la figura 5.4.

5.1.2. Segmentación

Tras haber realizado el filtro de color sobre la baliza, obtendremos todos los píxeles que la componen.

En experimentos con imágenes simuladas no se han encontrado problemas y la segmentación da resultados perfectos, quedando cada color de la baliza encerrada en una ventana de su tamaño (ver figura 5.3).

La segmentación con imágenes reales y simuladas en realidad es la misma, y su éxito o fracaso depende más de lo bien que se haya hecho el filtro de color.

La experiencia adquirida en estas pruebas es que la segmentación nunca resulta ser perfecta. La ventana hallada suele ser un poco más grande o más pequeña que el color que encierra, pero habitualmente bastante preciso. Generalmente influye el que la baliza sea redondeada o la propia inclinación de la cámara, ya que los segmentos hallados no pueden estar torcidos ni redondeados.

De lo bien que resulte este proceso se podrán obtener unos buenos resultados en la posterior triangulación, ya que dependerá de ellos.

En la figura 5.2 se han capturado imágenes correspondientes a dos experimentos. El primero de ellos (figura 5.2 arriba) muestra la segmentación de una baliza rosa y

azul y el segundo está segmentando dos balizas amarillas y rosas (figura 5.2 abajo) con ciertos problemas en la imagen izquierda, en la baliza izquierda.

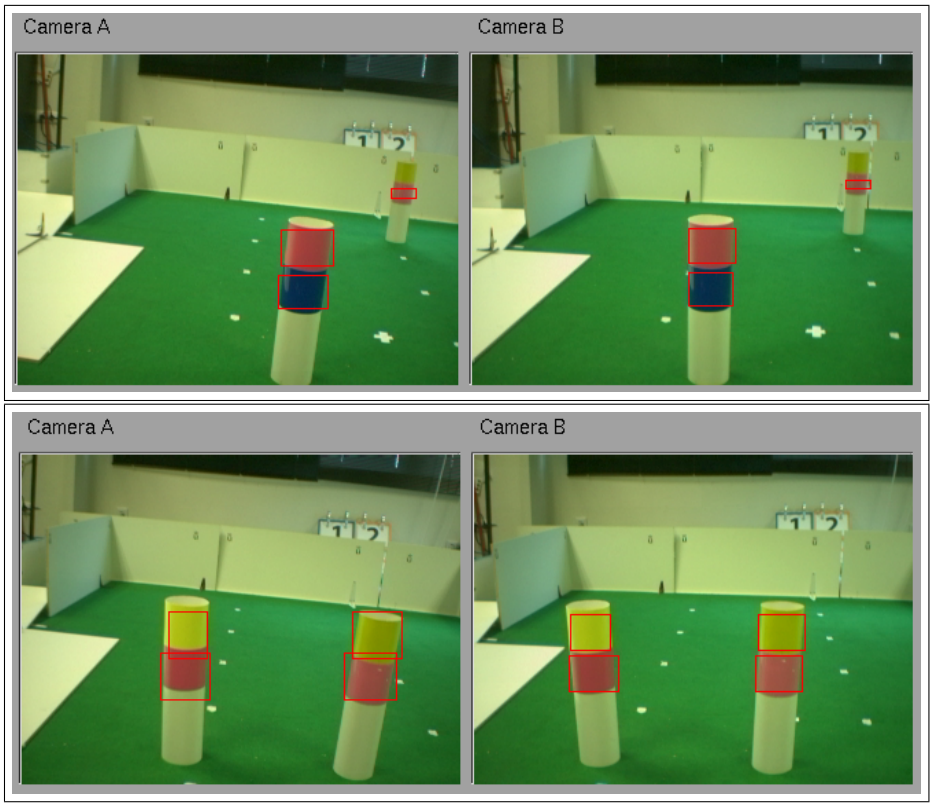

Figura 5.2: Dos imágenes en el proceso de segmentación de balizas.

5.1.3. Filtro de Bordes, Rectas y Segmentos

Análogamente a la segmentación de balizas, el filtrado de bordes se hace a la perfección en imágenes simuladas. En la obtención de rectas sobre estos bordes nos hemos encontrado diversos problemas. Como se comentó en 4.2.4, las rectas que se obtendrán tras el filtro, solo pueden tener ángulos enteros (por razones de eficiencia del algoritmo). Esto provoca que las rectas obtenidas no cuadren exactamente con las rectas reales que se ven en la imagen. Así, en algunos puntos se asemejan a la realidad, pero puede que en los extremos se desvíe de los bordes. La obtención de los segmentos de esas rectas depende de lo bien que se hayan obtenido las rectas. Así, cuanto más se haya ajustado el ángulo a la realidad, se tienen segmentos más precisos.

En general todo este proceso funciona bien con imágenes simuladas, sobre todo con los bordes del suelo que están a los lados del robot. En los bordes del suelo que son

paralelos al horizonte aparece una precisión menor, que no se ha conseguido solventar.

En la figura 5.3 se muestra cómo se ha segmentado la baliza virtual y los segmentos rectos del suelo detectados en el último paso. Se aprecia ligeramente en los segmentos rectos, que hay partes que están más pegadas al borde verde que otras, este es el problema comentado anteriormente. El esquema encargado de la segmentación pinta en rojo sobre las imágenes para mostrar los segmentos detectados.

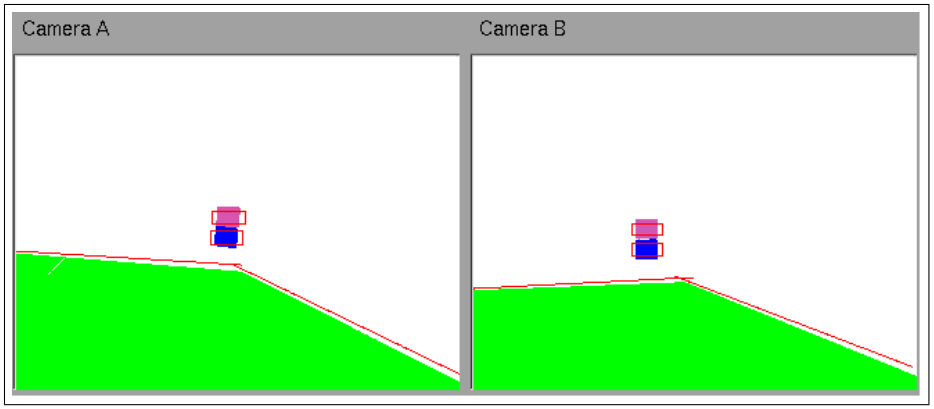

Figura 5.3: Imágenes simuladas en un momento del experimento sobre visión 2D.

La mayoría de los problemas encontrados con cámaras reales son los mismos que para las virtuales, pero incrementados debido al peor funcionamiento del filtro de color. Como ya se dijo, esta parte depende del filtro de color que se haga sobre el suelo para poder obtener los bordes. Para los experimentos en el pasillo hemos necesitado colocar unas pegatinas para facilitar este proceso.

En la siguiente imagen (figura 5.4) se muestra una captura de pantalla de las imágenes procesadas para obtener los segmentos rectos. Se aprecia en este caso que el segmento obtenido es bastante preciso. Al igual que en imágenes simuladas lo que peor detecta la aplicación son los bordes paralelos al horizonte.

5.2. Experimentos con Visión 3D

Esta sección está destinada a experimentar con la triangulación 3D de puntos (balizas) y segmentos (bordes del suelo). De estos resultados depende la correcta reconstrucción tridimensional de la escena. A su vez, dependen de la calidad de las segmentaciones obtenidas en las imágenes, ya probadas en la anterior sección.

Para la realización de los experimentos reales ha sido necesario calibrar las cámaras.

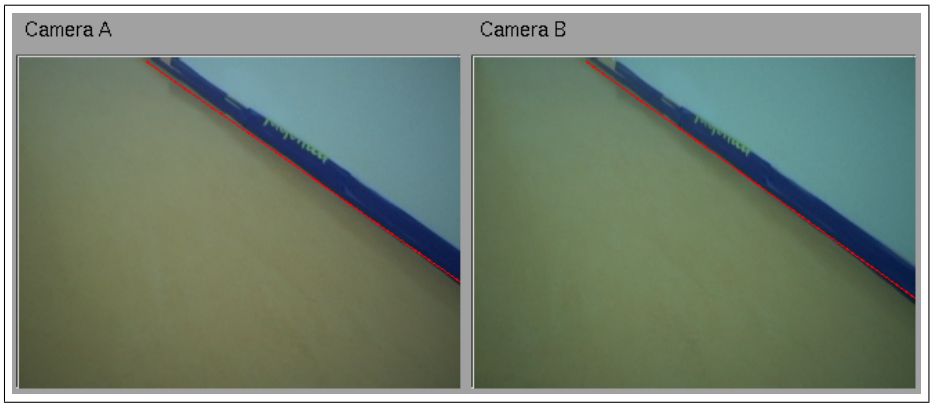

Figura 5.4: Imagen tomada de la interfaz experimentando con los segmentos del suelo.

Es de mención comentar de nuevo que sin esta calibración sería totalmente imposible realizar la triangulación 3D. Además, parte de las diferencias obtenidas entre utilizar imágenes simuladas (con una calibración perfecta) y reales, se debe a que es muy difícil calibrar exactamente estas últimas. Esto se debe a que la mínima desviación en la calibración hace que los resultados estén desviados, más cuanto más lejos esté el objeto. En nuestro caso, no se necesita una precisión milimétrica para navegar, así que el sistema funciona correctamente.

5.2.1. Calibración de Cámaras

Progeo se basa en un *modelo de cámara* para realizar las transformaciones, como ya se dijo en la sección 3.2.3, utiliza el *modelo Pinhole*. Este modelo tiene una serie parámetros que serán necesarios conocer para cada cámara en concreto, ya que *progeo* los utiliza para sus funciones.

El proceso de calibración consiste en obtener los *parámetros intrínsecos* y *parámetros extrínsecos* de las mismas. En el caso de los parámetros intrínsecos es necesario conocer la *distancia focal* y el tamaño del *píxel central* del plano imagen. Para los parámetros extrínsecos se necesita conocer la *posición 3D* de la cámara y la *orientación*.

Para obtener estos parámetros se ha utilizado software de ayuda. Los parámetros intrínsecos se obtuvieron con una biblioteca llamada *OpenCV*[1], gracias a la cual, se obtuvo la distancia focal y las coordenadas del píxel central (por el que pasa la recta focal)(ver tabla 5.2.1). Recordemos que las imágenes con las que se ha trabajado son

[1]http://sourceforge.net/projects/opencvlibrary/

Distancia Focal	Píxel Central (u_0, v_0)
411,15	(117,5, 157,5)

Cuadro 5.1: Parámetros intrínsecos para las cámaras *isight*

de 320x240, y al contrario de lo que cabría esperar, no se han obtenido las coordenadas $(120, 160)$ (para *progeo* la coordenada x corresponde con el eje vertical de la imagen), aunque sí próximas.

Una vez hallados los parámetros intrínsecos, se pueden averiguar los extrínsecos con la ayuda de la biblioteca *ARtoolkit*[2]. Para usarla se necesita imprimir un patrón en papel (figura 5.5 izquierda). Una vez fijado el patrón en un sitio a la vista de las cámaras, se arranca el programa (figura 5.5 derecha). Éste nos da la posición de la cámara respecto del centro del patrón y una matriz de rotación que nos permitirá obtener la inclinación de dicha cámara y un punto del foco de atención (a donde mira la cámara).

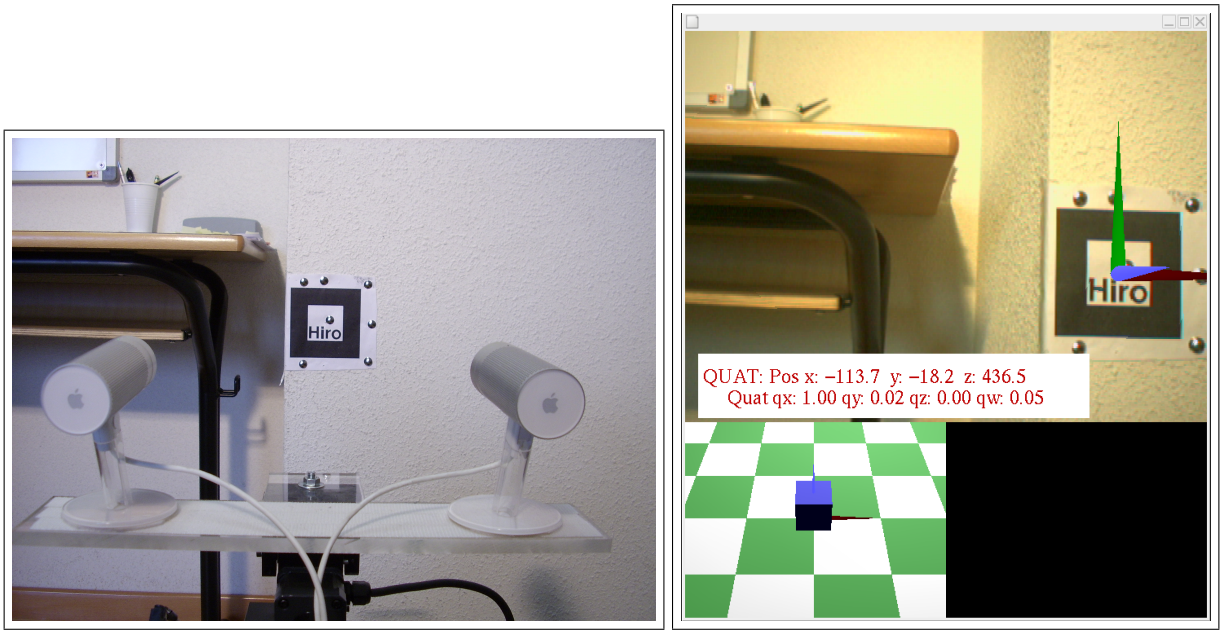

Figura 5.5: Patrón *Hiro* a vista de las cámaras e imagen del software *ARtoolkit*.

Una vez se consiguen los parámetros extrínsecos respecto del patrón, sólo queda transformarlos al sistema de referencia del robot (para saber en qué posición están respecto de él). Esto se hace midiendo en qué punto está el centro del patrón respecto del centro de masas del robot (punto elegido como centro) y realizando la traslación oportuna.

[2]www.hitl.washington.edu/artoolkit

5.2.2. Triangulación de Balizas

La triangulación de balizas en imágenes simuladas ha resultado ser muy precisa, del orden de centímetros. Debido a que la segmentación y a su vez el filtrado, fueron buenos y la estimación de la posición 2D de la baliza en cada cámara también, ha facilitado que las rectas de retroproyección crucen en el sitio adecuado. Una imagen donde se muestra la triangulación con balizas simuladas es en la figura 4.17.

Triangulación de una Baliza Real

En el primer experimento se colocó una baliza delante del robot y comprobaremos si sitúa adecuadamente su posición en la imagen de escena virtual. El escenario inicial se muestra en la figura 5.6. En él puede verse cómo las cámaras están mirando a una baliza de colores rosa y azul. La baliza está situada a unos dos metros de distancia delante suyo.

Figura 5.6: Escena real del experimento *Triangulación de una Baliza*.

Para visualizar mejor si se está identificando la baliza en cada imagen, se va a activar el esquema *segmentation*, que pintará en las imágenes los márgenes segmentados de la baliza. Luego se activa el esquema *project3d* que se encarga de realizar la proyección y dar una estimación de la posición de la baliza.

En la figura 5.7 se muestra la imagen de escena ampliada sacada de la interfaz en ese mismo momento.

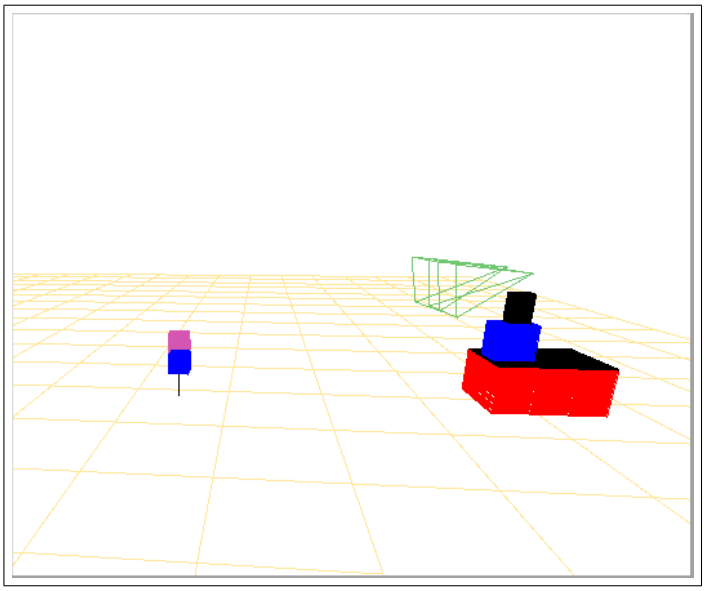

Figura 5.7: Escena real del experimento 5.2.2.

Se aprecia como la estimación de la posición de la baliza es correcta y se asemeja a la fotografía del escenario inicial (figura 5.6).

El balance general que puede hacerse con el funcionamiento de la triangulación de balizas en imágenes reales es que, aun con los errores de filtrado y calibración, se ha conseguido tener una precisión centimétrica. Sin embargo, esta precisión es bastante inferior a la obtenida con imágenes simuladas. Esta diferencia viene de nuevo heredada de que no se consigue una segmentación perfecta de los colores de la baliza y de los errores en la calibración de la cámara real. Así, el punto 2D dado de la baliza en cada imagen puede estar algo desviado. Esto provoca que las rectas de retroproyección corten en puntos más alejados o más próximos que el real.

En general, la coordenada y y z de la posición 3D de la baliza son correctas, y se aprecian desviaciones en la coordenada x, que se corresponde con la profundidad. Esta desviación es mayor cuanto más lejos se encuentre la baliza, pero es mínima cuando se encuentra a dos metros o menos. Por esta razón no es preocupante el error obtenido, ya que en el sistema final el robot se irá acercando a la baliza y actualizará su posición correctamente según se acerca.

Triangulación con dos Balizas Reales

El siguiente experimento con las balizas es semejante al anterior, pero se colocaron dos balizas en el campo visual del robot. Con ello se comprueba que el

emparejamiento es correcto y que la triangulación funciona con más de un objeto simultáneamente.

El escenario inicial ahora es el mostrado en la figura 5.8.

Figura 5.8: Escena real del experimento *triangulacion con dos balizas*.

Se activan de nuevo los esquemas *segmentation* y *project3d*. Veremos si se consiguen superar los objetivos del experimento, realizando adecuadamente la correspondencia y situando ambas balizas en la escena 3d.

Como se aprecia en la figura 5.9, se ha reconstruido adecuadamente la parte de la escena que es de nuestro interés. Ambas balizas quedan situadas en el mundo 3d virtual a una misma distancia del robot y paralelas a este (como en la realidad).

La correspondencia entre balizas en la imagen se resuelve adecuadamente en imágenes reales, pero para ello se han tenido que ampliar los umbrales que pedíamos en el anterior capítulo (sección 4.3.1). En concreto el umbral de la distancia a la línea epipolar. Esto se debe a que a veces la segmentación no es perfecta, lo que hace que el cálculo de su centro en una de las imágenes sea distinto al de la otra, lo que desvía la línea epipolar ligeramente.

5.2.3. Triangulación de Segmentos del Suelo

Siguiendo la línea de los experimentos anteriores, se pone a prueba el mecanismo de detección y triangulación de bordes del suelo. Estos experimentos

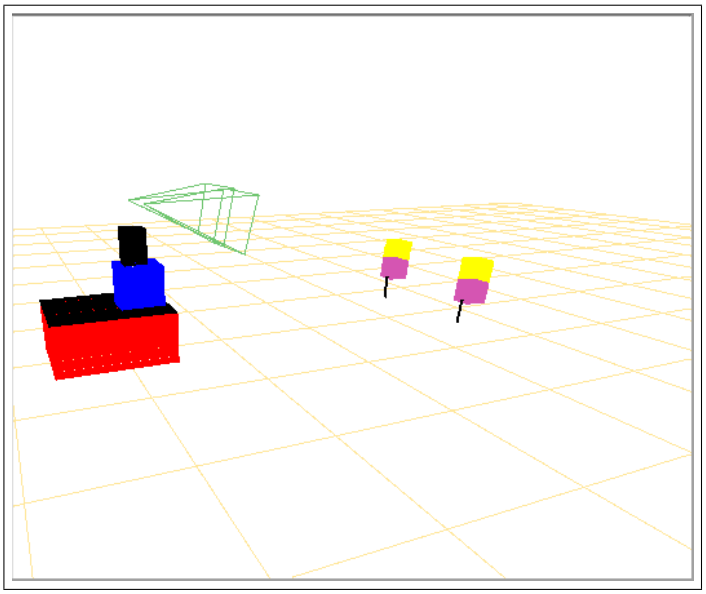

Figura 5.9: Imagen de la escena reconstruida en la situación de la figura 5.8.

dependen en gran parte de los segmentos 2D obtenidos anteriormente.

En los experimentos con imágenes simuladas, como se verá en la sección 5.4.1, la estimación de los bordes laterales al robot es muy precisa, con errores del orden de centímetros. También funciona adecuadamente para segmentos paralelos al horizonte (en frente del robot) siempre que no estén lejos.

Para posiciones lejanas de estos segmentos se obtienen ciertas desviaciones en sus ángulos. Esto es debido a que el filtrado de rectas no es perfecto y se hereda este problema ya comentado anteriormente.

De nuevo no es preocupante que estos segmentos estén algo desviados en la lejanía, ya que cuando el robot navegue en el sistema final, actualizará su posición al acercarse.

En el siguiente experimento se prueba este funcionamiento con cámaras reales. Para ello se ha colocado el robot en el pasillo del departamental de la escuela y se ha situado el cuello de forma que pueda ver los límites del suelo.

El escenario propuesto puede verse en la figura 5.10 y se prueba si se reconstruyen bien los bordes del suelo en la imagen de escena. Puede verse que se han colocado pegatinas en el rodapiés para facilitar el filtrado de bordes y segmentos.

Para este experimento se activa el esquema *segmentation* para comprobar si se efectúa correctamente el análisis de segmentos rectos sobre el suelo. Por último, se activa *project3d* para obtener la representación 3d.

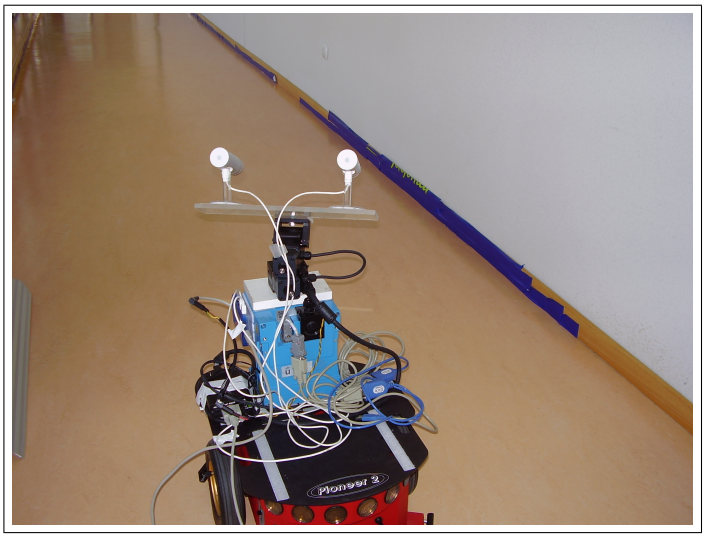

Figura 5.10: Escena para probar la triangulación del suelo.

En la figura 5.11 se aprecia la imagen virtual de escena reconstruida durante la ejecución. En ella puede verse cómo se han representado correctamente los bordes del suelo que está viendo, asemejándose a la foto real tomada en 5.10.

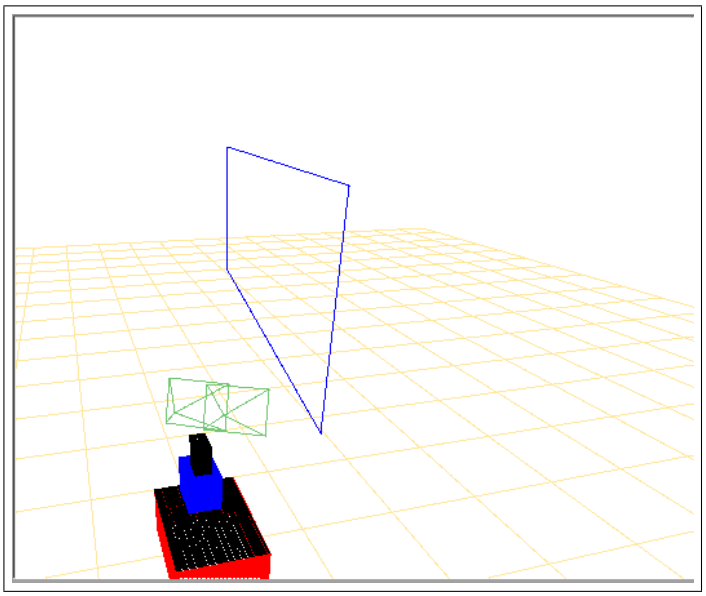

Figura 5.11: Ampliación de la escena virtual reconstruida.

Análogamente a los resultados obtenidos en imágenes simuladas, en la realidad ha habido falta de precisión para los bordes que están en frente del robot, paralelos al horizonte. No así para los bordes laterales del suelo, en los que se consigue una precisión

centimétrica.

El objetivo de obtener esta triangulación 3D, es que el robot sea capaz de navegar sin chocarse con las paredes. Para este fin es necesario que la triangulación sea buena. Experimentos posteriores probarán si la precisión obtenida es suficiente para la navegación.

5.3. Experimentos con el Sistema de Atención

Los experimentos siguientes ponen a prueba el control de atención en la escena y su reconstrucción 3D. Primero se ha realizado un experimento solo con balizas y posteriormente uno que prueba el sistema de atención con balizas y suelo.

5.3.1. Sistema de Atención con Balizas

El primer experimento consiste en colocar balizas por el escenario de prueba y comprobar si el sistema es capaz de encontrarlas todas realizando un seguimiento sobre ellas.

Los experimentos realizados en simulación han funcionado correctamente. Las dinámicas de vida y saliencia han conseguido que cuando las balizas se localizan, se visitan de nuevo constantemente, actualizando su posición si se movía alguna de ellas (ver experimento 5.4.1 que integra esta parte).

Después de que la triangulación de balizas con imágenes simuladas fuera muy precisa, ahora se obtiene la misma precisión. Parece algo normal, pero hay que tener en cuenta que las cámaras ahora se mueven constantemente y podría darse el caso de que la triangulación se hiciera un poco después o antes de tener las imágenes quietas, dando lugar a errores.

En cuanto a balizas reales, hemos llevado a cabo un experimento en el que se han colocado varias balizas reales alrededor del robot, de manera que no es capaz de verlas en el campo visual de una imagen.

La situación inicial de las balizas alrededor del robot real se aprecia en la figura 5.12. En ella puede verse que se ha dispuesto de tres balizas, dos amarillo-rosa y una rosa-azul. En este experimento se buscarán las balizas amarillo-rosa y también se busca probar que no añade la baliza que no es del tipo de las anteriores.

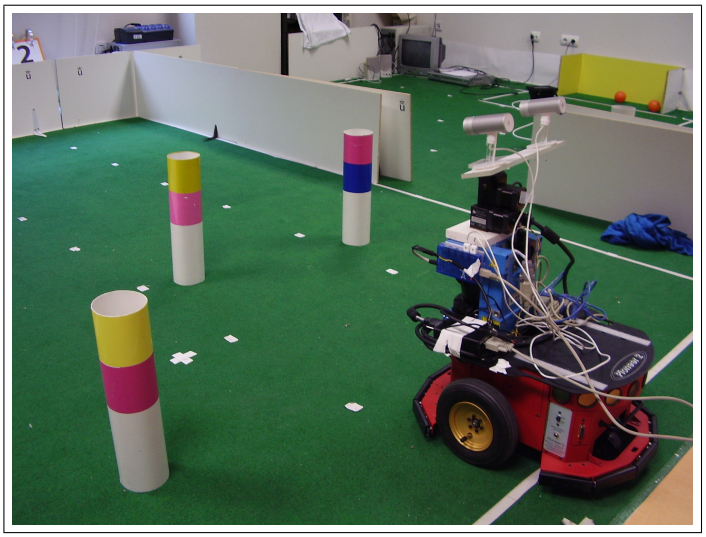

Figura 5.12: Escenario con balizas separadas.

En este caso solamente se activa el esquema *attention3d* (que ya engloba todos los pasos de filtrado, segmentación, etc.) para tener al máximo los recursos computacionales.

La secuencia de imágenes mostrada en la figura 5.13, ilustra como el robot va haciendo el barrido exploratorio. Al principio la escena virtual aparece vacía (**1**). Más tarde encuentra una baliza y la sitúa en la imagen virtual (**2**). Así hasta encontrar la segunda baliza (**3**). Cuando ha encontrado ambas, reparte su atención entre ellas y continúa con el barrido (**4**). En la última imagen de la secuencia (**5**), hemos movido la baliza izquierda para comprobar si actualizaba su posición cuando la volviera a visitar y el resultado ha sido positivo. Puede verse el escenario real final en la imagen 5.14.

El balance obtenido tras estos experimentos es que el sistema de atención funciona a la perfección. Los objetos se identifican y se insertan adecuadamente en el control de atención 3D. Cuando se visitan de nuevo, se miran exactamente de modo que queden centrados en la imagen, también se actualiza su posición cuando se desplazan y por último se olvidan cuando llevan un tiempo sin aparecer en la escena.

El principal problema encontrado ha sido debido al cambio de iluminación que sufren las cámaras cuando se mueven.

Este problema ha repercutido en tener que retocar el algoritmo de atención. En concreto se ha tenido que introducir un tiempo de espera cada vez que se gira el cuello mecánico, para que las cámaras tengan tiempo de ajustar automáticamente su

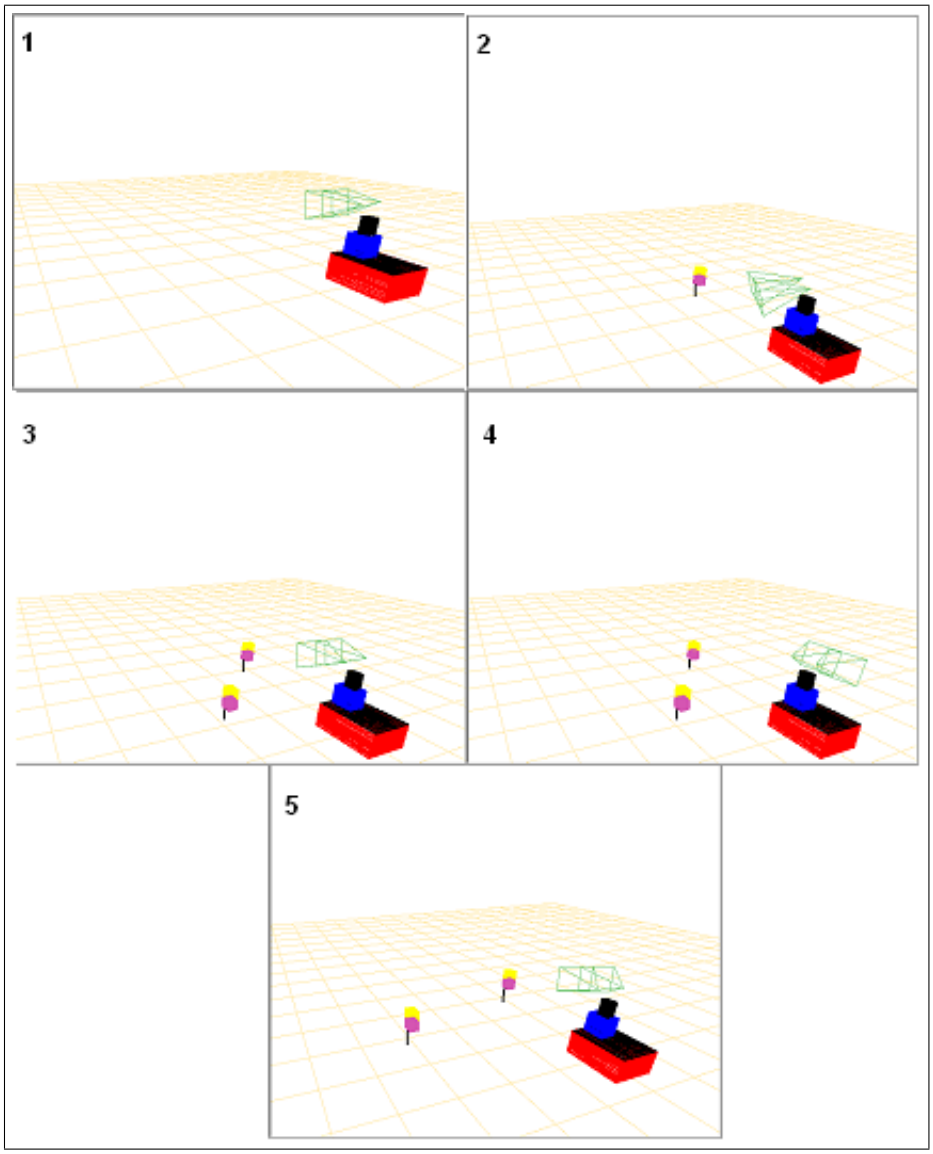

Figura 5.13: Secuencia de imágenes del escenario virtual, ejecutando el experimento con balizas separadas.

iluminación. En concreto en estas pruebas, la espera introducida ha sido de un segundo. Hemos constatado que esperando este tiempo para procesar la imagen (desde que el cuello se paró) es suficiente para realizar un buen filtrado de color.

A lo largo de los experimentos se ha comprobado que si se realiza una mala estimación de la posición de una baliza en un determinado momento, no causará mayores problemas. Esto se debe a que las balizas, como los bordes del suelo, tienen asignada una determinada vida. Para que el objeto no muera, debe refrescarse su vida periódicamente siendo visitado, pero si la estimación ha sido errónea, cuando se mire a ese punto para renovar su vida no se verá nada y las balizas *malas* acaban

Figura 5.14: Escenario real tras alejar la baliza izquierda del robot.

muriendo (igual ocurre para los segmentos del suelo).

5.3.2. Sistema de Atención con Balizas y Suelo

Este experimento está enfocado a comprobar la integración de los segmentos 3D en el sistema de atención, y ver si casa bien con lo ya probado con las balizas.

En el caso de las imágenes simuladas, al igual que el sistema de atención con balizas simuladas, la integración de los segmentos del suelo se ha realizado correctamente (ver experimento 5.4.1 que integra esta parte).

Estos segmentos se han descubierto sin mayor problema, al igual que las balizas y el control sobre ellos ha producido algunos resultados interesantes. Ya se explicó en el capítulo 3 que el punto de atención escogido para cada segmento, será aquel que corte con las rectas de 50^o y -50^o respecto del robot, y si no cortan, se escogerá el punto medio. Con este funcionamiento, la atención refresca perfectamente los segmentos laterales, y frontales. Los laterales se actualizan cuando se mira a ese punto de corte, y los frontales porque se mira a su punto medio.

Un problema encontrado es con los segmentos que detecta detrás del robot, ya que estos no cortan las rectas, y se pone como punto de atención el punto medio del segmento. En las pruebas realizadas se ha simulado un pasillo que tiene fin delante y detrás, y cuando detectaba parte de el de detrás, más tarde intentaba mirar a su punto medio, pero el movimiento de la pantilt no posibilita un giro total hacia detrás, así que el sistema no era capaz de refrescar la vida de este segmento y acaba muriendo.

Visto con perspectiva, no supone un problema, ya que al robot le interesará más lo que está delante y a los lados para poder navegar.

En cuanto a imágenes reales, se ha realizado un experimento en el que se ha preparado un escenario con un pasillo artificial en el laboratorio de robótica, y se ha colocado el robot en medio. Además se ha dispuesto una baliza en frente suyo (figura 5.15).

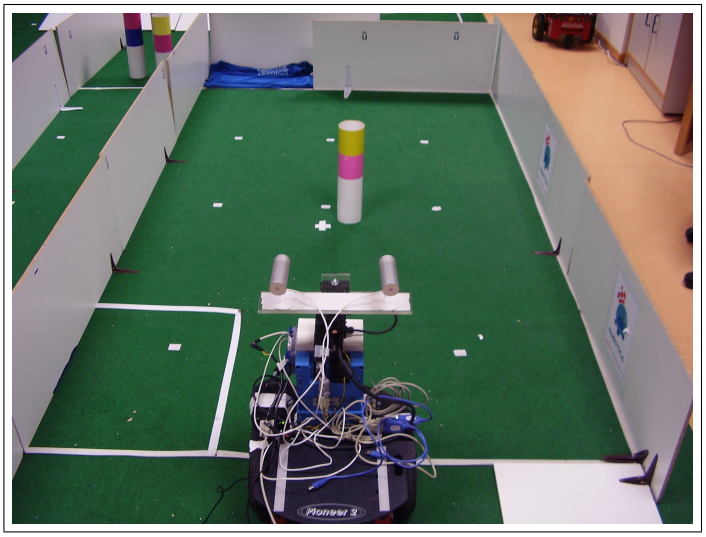

Figura 5.15: Escenario propuesto para probar la atención de baliza y suelo.

De nuevo, sólo se activa el esquema *attention3d*, ya que no se necesita ver que hace correctamente el filtrado de segmentos suelo.

La secuencia de la figura 5.16 muestra como ha sido la prueba. Partiendo del mundo vacío (**1**), primeramente se ha detectado la parte cercana derecha del pasillo (**2**). En la imagen (**3**) ha descubierto la prolongación de dicho borde, haciendo la correspondencia entre segmentos, y lo ha unido al anterior. Siguiendo con el barrido, descubre la baliza (**4**). Más tarde, en (**5**), reconstruye la parte del fondo del pasillo creado. Mientras continúa el barrido exploratorio, visita los segmentos hallados y la baliza, mirando a la derecha y delante del robot. En (**6**) comienza a reconstruir la parte izquierda del pasillo y lo primero que ha visto es su parte central. En esta imagen puede apreciarse cómo también ha reconocido la línea blanca perpendicular al pasillo como segmento frontera del suelo. Continuando con el barrido, detecta el comienzo del pasillo izquierdo (**7**) y, finalmente, en (**8**) reconstruye y empareja la parte final de dicho borde.

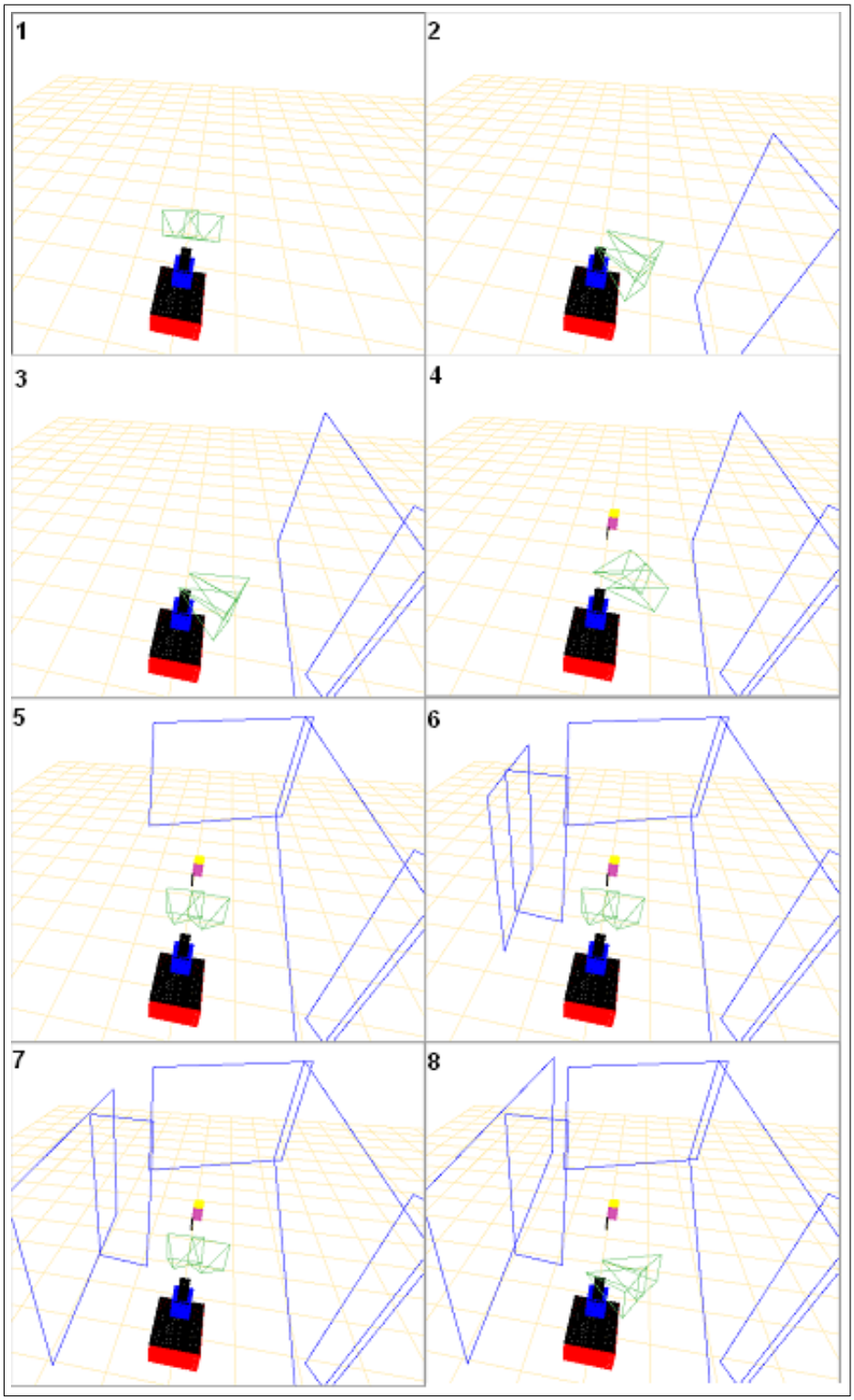

Figura 5.16: Secuencia de imágenes obtenidas durante la ejecución de la aplicación.

La imagen de escena final ampliada está en la figura 5.17, semejante a la realidad vista en la imagen 5.15. Con este experimento concluyen las pruebas para el sistema

de atención con balizas y bordes del suelo con el robot estático.

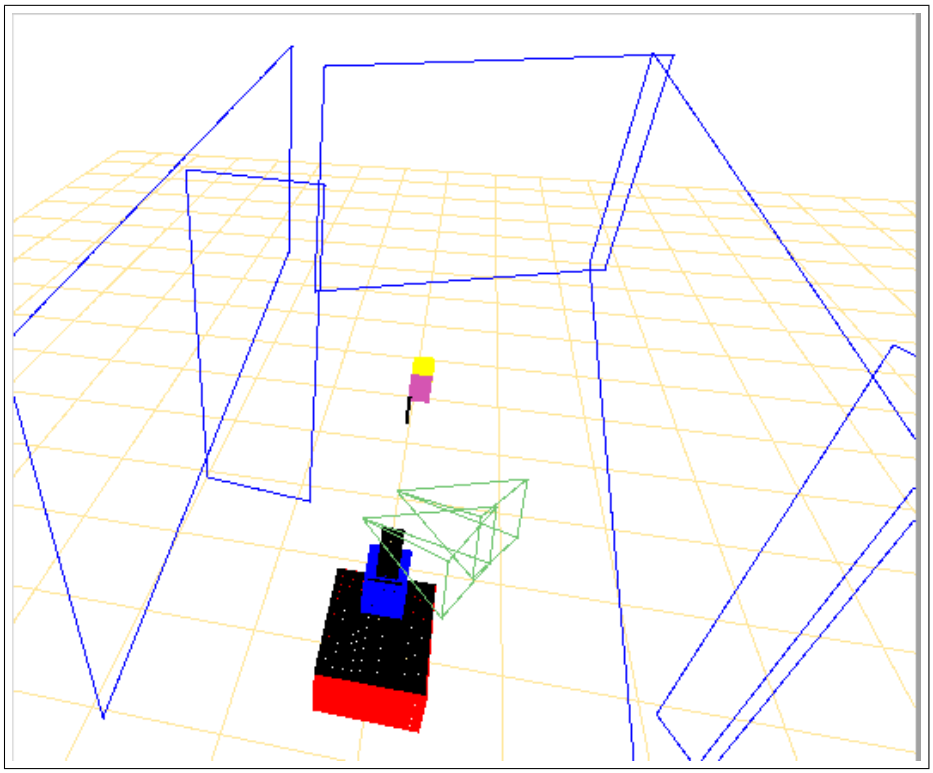

Figura 5.17: Imagen de escena reconstruida tras el experimento *Sistema de Atención con Balizas y Suelo.*

Las conclusiones adquiridas tras esta prueba experimental es que el sistema reconoce bien las partes del pasillo laterales, y las cercanas. Pero, tal y como se aprecia en la imagen 5.17, el final del pasillo es reconocido con una cierta desviación en su ángulo. En la foto del escenario real, se aprecia que al final del pasillo (parte izquierda) se tuvo que colocar una camiseta, ya que no cabía un nuevo tablón blanco. Esto ha podido incurrir en un cierto error en la obtención del ángulo de las rectas, lo que ha llegado a la reconstrucción final. Este final de pasillo está situado a unos cinco metros de distancia, lo que no resulta preocupante para la futura navegación, ya que cuando se aproxime a él mejorará mucho su estimación.

Es de destacar, que al final del experimento se ha mantenido el límite encontrado por la pequeña raya blanca sobre el césped verde.

Con este experimento se ha probado el correcto funcionamiento de la identificación y triangulación 3D de segmentos rectos del suelo, y su integración en el sistema de atención y con las balizas.

5.4. Experimentos de Integración Final

Los últimos experimentos realizados probarán el sistema final con una ejecución típica de la aplicación. En ellos se pone a prueba la integración de todos los subsistemas. Primero se realiza el experimento en simulación, y para finalizar, con el robot real.

5.4.1. Experimento Final con Imágenes Simuladas

Como ya se ha comentado, el sistema se ha ido probando primeramente con imágenes simuladas. Por esta razón, se muestra en esta sección el funcionamiento final del sistema con este tipo de imágenes. Al funcionar en modo simulación se consiguen tener unos filtros de color y bordes, y unas segmentaciones sin ruido alguno y, lo que es más importante, una calibración de cámaras perfecta.

En la escena simulada se ha dibujado un pasillo de color verde, que representará un pasillo de la realidad. También se han colocado dos balizas, una de cada tipo. La baliza de *rosa-azul* (la primera que buscará el sistema de atención) se sitúa al final de este pasillo, mientras que la otra se coloca al principio. Puede verse esta representación en la imagen 5.18.

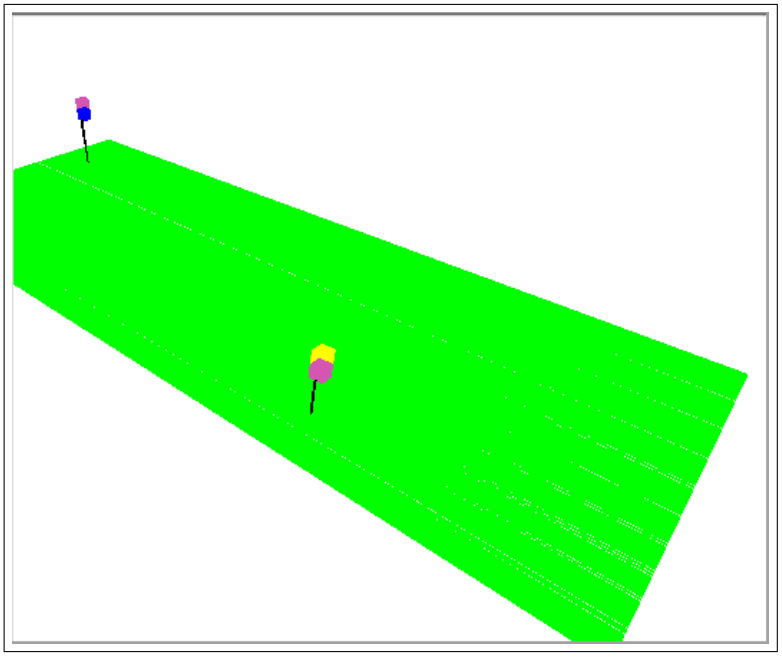

Figura 5.18: Escena inicial simulada para el experimento 5.4.1.

Inicialmente se activa el esquema de atención y automáticamente se activará también el esquema de navegación cuando sea oportuno. Como se trabaja en modo simulación, lanzaremos el simulador *Stage* en mundo vacío sin obstáculos, que se encargará únicamente de dar los datos odométricos y mover el robot cuando se le comande. También será necesario activar el esquema simulador de pan-tilt.

La secuencia de imágenes de la escena reconstruida, puede verse en la figura 5.19. En (**1**) comienza sin ningún conocimiento de lo que le rodea. En (**2**) y (**3**) reconstruye parte del borde derecho del pasillo y en (**4**) parte del izquierdo. (**5**) muestra cómo se tiene casi la totalidad del borde izquierdo reconstruido e incluso una parte de detrás. Al

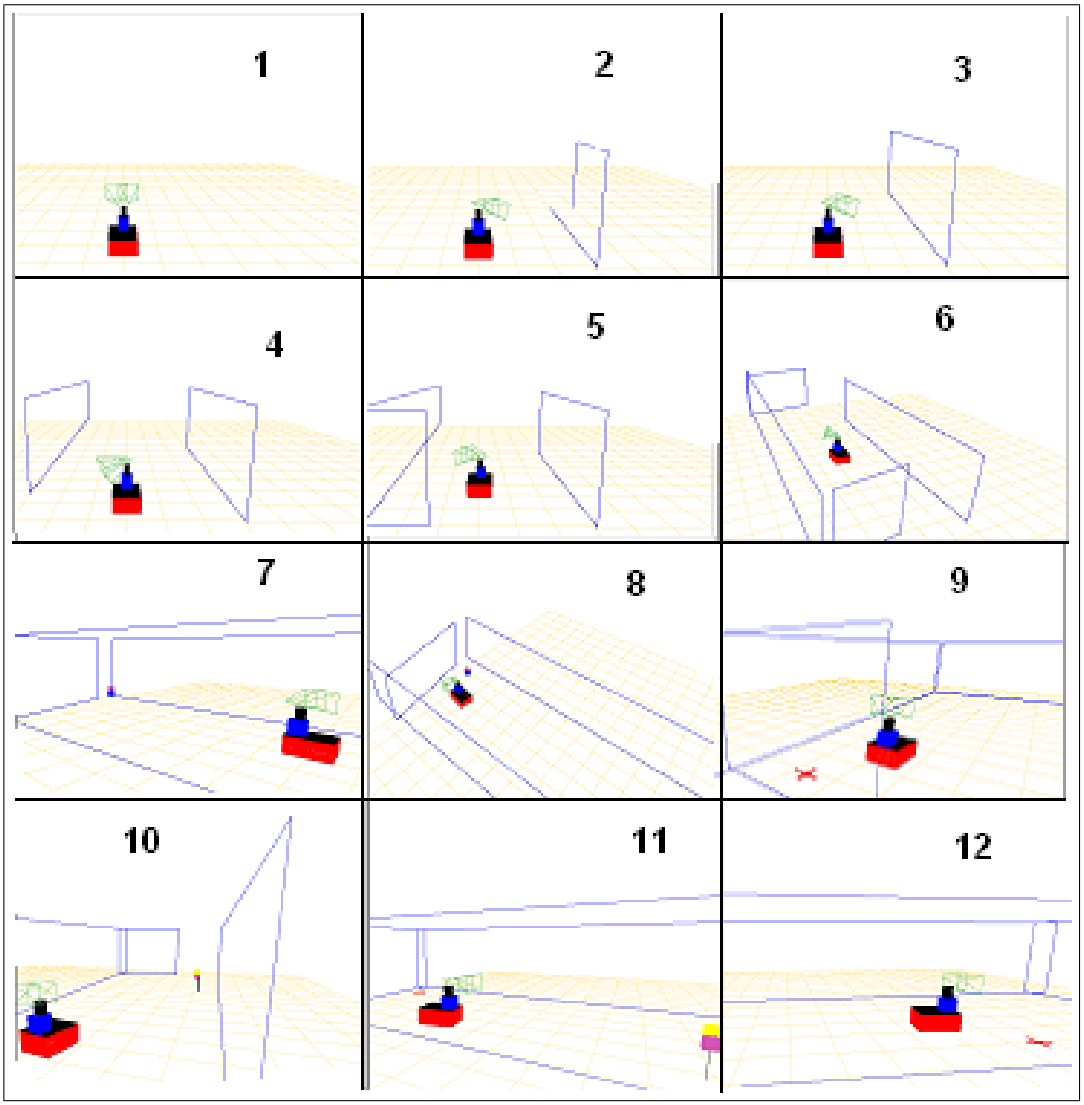

Figura 5.19: Secuencia de ejecución de la aplicación con imágenes virtuales.

terminar el barrido exploratorio se activa el esquema *vff* para que el robot navegue. En

esta ocasión no ha encontrado todavía ninguna baliza, ya que ésta se situó lo bastante lejos como para no ser vista desde la posición inicial. Así que el esquema ha calculado un punto de exploración seguro y avanza hacia el (paso **6**). En este punto comienza de nuevo el sistema de atención, donde se ve que ha reconocido parte del final del pasillo. Por fin, en (**7**) se localiza la baliza y esta vez sí, cuando se activa la navegación, se dirige hacia ella (**8**).

Cuando se alcanza el objetivo se marca en el suelo una cruz roja (**9**), y es cuando se cambia automáticamente el modo atención. A partir de ese momento, pasa de buscar balizas *rosa-azul* a buscarlas *amarillo-rosa*.

Esta vez sí reconoce la baliza desde su posición, así que no necesitará deambular. En el paso número (**10**) se ve como ha detectado la nueva baliza. Por último, en el paso (**11**) se encamina hacia la baliza y en (**12**) llega a ella.

Al término de este experimento se puede concluir que se han cumplido los objetivos marcados inicialmente, en mundos simulados. El robot ha sido capaz de reconstruir las partes interesantes de su entorno y además ha sido capaz de navegar por él. Ha encontrado una baliza que no estaba en su campo de visión inicialmente y ha cambiado el modo de atención correctamente cuando alcanzó su objetivo.

En cuanto a los requisitos impuestos al comienzo, se pedía que el sistema fuera vivaz y eficiente. Hay que comentar, que cuando hemos trabajado con imágenes simuladas, la velocidad de refresco de la interfaz no es muy rápida (unas 4 iteraciones por segundo) Esto se debe a que el esquema encargado de la interfaz gráfica, debe realizar sus tareas típicas más la de generar las dos imágenes virtuales, lo que significa mucha carga. Esto no ha condicionado que el sistema global se comportase eficientemente, consiguiendo que el robot navegue a $200\frac{mm}{s}$.

5.4.2. Experimento Final con Imágenes Reales

En este experimento se prueba el sistema al completo funcionando en el robot real.

En el escenario propuesto para la prueba se sitúa el robot en medio del pasillo. Además hemos colocado dos balizas, una de cada tipo. La primera la situamos en frente del robot, en medio del pasillo. Y la segunda se ha puesto cerca del robot para que, en caso de salir bien, vuelva a su posición inicial después de encontrar la baliza *rosa-azul*. Las fotos del experimento pueden verse en la figura 5.20.

En los primeros pasos se ve como hace el barrido exploratorio durante el cual está detectando los bordes del suelo (**2-4**). Esto se sabe porque a continuación, vuelve a mirar varias veces a sus lados, para refrescar la posición de los segmentos. Esta conducta se aprecia en las siguientes dos imágenes de la secuencia (**6-7**), justo antes de encontrar la baliza (**8**). El barrido exploratorio continúa mientras alterna la mirada entre los bordes de sus lados y la baliza. Al terminar el barrido comienza a navegar dirigiéndose hacia la baliza detectada (**10**). Al llegar a ella se aprecia que la baliza que está frente al robot ya no es de su interés. En cambio sí sigue mirando los bordes del suelo antes detectados, de manera que mantiene la escena reconstruida (**12-13**). Mientras continúa con esta conducta comienza un nuevo barrido en busca de balizas *amarillo-rosa*. Justo al final del barrido la encuentra y comienza de nuevo a navegar hasta que llega a ella (**14-18**).

Con este experimento se ha probado que el sistema final con robot real y cámaras reales funciona. Hemos comprobado que se ha integrado lo realizado en etapas anteriores, filtrando los colores de balizas y suelo, segmentando los colores de la baliza, identificando los bordes del suelo, triangulando balizas y segmentos rectos, etc. Sin el buen funcionamiento de todas las partes de la aplicación, este comportamiento de navegación no sería posible.

La principal conclusión obtenida tras todos los experimentos es que el sistema es muy sensible a la luz. Esto es normal ya que las cámaras son el único sensor de entorno del que dispone la aplicación y si no se obtiene buena información de ellas, el sistema fracasará. Esto supone, principalmente, que los primeros filtros de color hechos a las imágenes son la base para que el resto funcione adecuadamente. Si se consigue tener bien filtrado los colores de las balizas (no muy difícil) y del suelo (bastante difícil en el pasillo) se comprueba que el sistema funciona adecuadamente y además a una velocidad vivaz.

Figura 5.20: Secuencia de ejecución típica de la aplicación.

Capítulo 6

Conclusiones y Trabajo Futuro

En anteriores capítulos se explicó el objetivo principal de este proyecto. Tras ello, se describió el diseño y la implementación de la solución realizada y las pruebas que la validan experimentalmente. En este capítulo se presentan las conclusiones obtenidas, las dificultades principales encontradas y las líneas por las que se podría mejorar su funcionamiento futuro.

6.1. Conclusiones

Los objetivos marcados inicialmente para este proyecto fin de carrera, implicaban que la aplicación fuera capaz de dotar al robot de un sistema de atención en tres dimensiones, mediante el uso de dos cámaras y un cuello mecánico. Ésta atención tenía el fin de reconocer las partes interesantes de todo lo que rodeara al robot. En este caso, se consideró importante ser capaz de identificar la frontera del suelo y reconstruirla en 3D para que el robot pudiera navegar por el *mundo*. También se consideró importante reconocer una serie de balizas repartidas por dicho *mundo*, proporcionando un objetivo al robot cuando navegase con ellas y comprobar el funcionamiento del control de atención. Por último se pretendía probar todo el sistema en su conjunto.

La conclusión general después de haber acabado es que se han cumplido satisfactoriamente los objetivos marcados. El sistema de atención ha funcionado perfectamente y ha resuelto el problema de cómo mantener la escena actualizada atentivamente. Las balizas y el suelo se han identificado correctamente y la estimación de su posición 3D ha posibilitado una correcta navegación.

A continuación se comenta en qué medida se han satisfecho los subobjetivos marcados en el capítulo 2:

1. *"Procesamiento de Imágenes Monoculares"*. Se propuso que la cámara fuera capturando imágenes continuamente para conocer la información del entorno en el que se situaba.

   a) *"Capturar Imágenes"*. El sistema debía ser capaz de realizar una toma de imágenes mediante cámaras firewire. Esto se consiguió desde un principio sin mayor problema, ya que lo resuelve la plataforma software usada.

   b) *"Distinguir objetos relevantes de los no relevantes en una imagen"*. Se exigía que las imágenes capturadas fueran analizadas mediante técnicas propias de visión computacional: filtrado de color, filtrado de bordes y segmentación 2D, para identificar los objetos relevantes (las balizas y la frontera del suelo).

   En los experimentos ha quedado suficientemente probado este apartado. La identificación de balizas se ha implementado usando un filtro de color HSI y más tarde realizado una segmentación recursiva en la imagen para agrupar los píxeles del mismo color en segmentos. Para el reconocimiento de los límites del suelo se ha realizado un filtro de bordes, la obtención de las rectas que contienen esos bordes y la discriminación en ellas de los segmentos del suelo en la imagen.

2. *"Localización 3D de Objetos"*. La aplicación debía ser capaz de localizar la posición tridimensional de objetos con la única ayuda de dos cámaras, mediante técnicas geométricas. Se pretendía que emparejara correctamente los objetos vistos en cada cámara y triangulara desde cada una para calcular su posición 3D. A su vez se pedía que se visualizara una representación 3D de la escena generando una imagen virtual.

   La localización de balizas y su triangulación 3D se ha visto probada en los experimentos (5.2.2), al igual que la triangulación de segmentos. La triangulación de balizas se ha conseguido gracias a la obtención de las rectas de retroproyección, que cortaban cada una a la baliza, hallando posteriormente el punto de cruce de ambas. La triangulación de segmentos se ha hecho mediante la suposición que los segmentos vistos en la imagen estuvieran en el plano $z = 0$ obteniéndose segmentos 3D, luego se descartaban si los calculados por cada cámara no emparejaban bien.

   También se ha mostrado cómo el sistema es capaz de emparejar adecuadamente las balizas y los segmentos del suelo usando diferentes técnicas para cada caso. En el caso de las balizas se han usado las rectas epipolares, tamaños y que la

zona de la imagen donde aparecen sea parecida. Para los segmentos se comparan los ángulos de cada uno y la distancia de los extremos a la recta de prolongación del segmento pareja.

La representación de la escena 3D en la interfaz gráfica ha satisfecho su misión, la de dar más vistosidad a la aplicación y ayudar a la depuración.

3. *"Control de Atención"*. En este subobjetivo se pedía la realización de un control de atención del cuello mecánico de modo que siguiese los diferentes objetos identificados (balizas o bordes suelo) en el *mundo 3D*. Su misión era la de ampliar el rango de visión a todo lo que hubiera alrededor del robot, aunque caiga fuera del campo visual instantáneo del par estéreo de cámaras. Para ello se necesitaba mover las cámaras y, por consiguiente, utilizar dicho cuello. Las características principales que se pedían eran:

   a) *"Reconocer nuevos objetos (balizas o bordes suelo)"*.

   b) *"Realizar un seguimiento sobre los diferentes objetos ya encontrados"*.

   c) *"Explorar zonas en busca de nuevos objetos"*.

   Todo esto ha quedado satisfecho con las pruebas, donde se ha visto que el sistema hace un barrido exploratorio adecuado. Cuando una baliza por ejemplo, es reconocida, se añade al sistema de atención y se visita periódicamente para comprobar nuevamente su situación. Se ha probado que si su posición cambia, el sistema la actualiza. Además el control de atención ha funcionado correctamente con las balizas y el suelo simultáneamente, siendo capaz de atender puntos 3D y segmentos 3D.

4. *"Navegación Visual Autónoma"*. Fue el último de los subobjetivos marcados y suponía la integración de los anteriores subsistemas. Además consistía en que el robot pioneer siguiera las *balizas-objetivo* a la vez que evitaba chocar con las paredes.

   Con los experimentos finales se pretendía probar este objetivo. Se ha conseguido superar con éxito y ver cómo el robot deambulaba cuando no encontraba una baliza. Cuando el sistema de atención la localizaba se marcaba como objetivo e iba hacia ella. También se ha puesto a prueba el algoritmo de cambio de objetivo, donde cada vez el objetivo era una baliza de colores distintos. Todo ello se ha cumplido mientras evitaba chocar con las paredes.

Además de los subobjetivos, se han cumplido los requisitos descritos en la sección 2.2. Estos eran:

1. El sistema debía ser lo más robusto posible, especialmente tolerante frente a cambios de iluminación en el entorno. Este requisito se ha satisfecho a medias ya que se ha conseguido que funcione la aplicación en un entorno controlado, pero en entornos con distinta iluminación deberán reconfigurarse los filtros de color.

2. Se pedía que los algoritmos fueran eficientes, para tener un sistema vivaz. Los principales esquemas se ejecutan a una media de 8 iteraciones por segundo y la captura de imágenes a $25 - 30 fps$. Estas velocidades han posibilitado una navegación vivaz en la que el robot se mueve a unos $200 \frac{mm}{s}$.

3. Las cámaras debían estar calibradas con precisión, para conocer sus parámetros y así poder realizar los cálculos geométricos necesarios. Se ha realizado una calibración con precisión suficiente para un correcto funcionamiento de la ejecución típica.

4. El diseño se ha concebido siguiendo la arquitectura *JDE* (sección 3.2.2). De este modo se ha implementado la aplicación siguiendo la arquitectura en esquemas, adaptando algunos esquemas de otros proyectos, y creando otros nuevos. Todo para dar lugar al diseño global expuesto en la sección 4.1

Este proyecto de ingeniería ha sido muy completo. Al principio se tenía un problema abierto del que no se sabía si se conseguirían obtener resultados positivos. Se ha realizado un diseño preliminar, una posterior implementación en esquemas y por último una serie de experimentos.

El sistema ha sido validado experimentalmente como se muestra en el capítulo 5. Se han realizado diferentes pruebas al prototipo realizado, primero en simulación y luego con el robot real. Se ha comprobado en ellas cómo el sistema iba cumpliendo los objetivos marcados inicialmente.

La superación de los objetivos marcados han supuesto un reto con imágenes simuladas y conseguir ajustar los algoritmos para que funcionen correctamente en el robot real ha supuesto un esfuerzo adicional, ya que aparecieron muchos problemas añadidos.

El sistema realizado incorpora componentes software ya existentes de otros proyectos, lo que a veces ha resultado difícil y requerido de adaptaciones a nuestras

necesidades concretas. El filtro de color, la segmentación de color [Hidalgo, 2006], la extracción de rectas [Peña, 2004] y las dinámicas de atención [Martínez, 2005] son ejemplos de ello. Sobre esos antecedentes hemos desarrollado mejoras significativas que han llevado a una funcionalidad completamente nueva.

Son aportes genuinos de este proyecto:

1. La extracción de segmentos borde desde rectas borde.
2. Los algoritmos de emparejamiento de balizas y segmentos suelo entre las imágenes del par estéreo.
3. El algoritmo de triangulación y localización del punto de corte entre las dos rectas de retroproyección y la triangulación de segmentos suelo.
4. El paso a 3D del algoritmo de atención:
   a) Relacionando las posiciones 3D con la posición adecuada del cuello mecánico y viceversa.
   b) Exploración de escena 3D.
   c) La introducción de los segmentos del suelo como puntos de atención, además de las balizas puntuales.
5. Integración de la navegación del robot con el algoritmo atentivo de reconstrucción de escena. El robot se mueve mientras percibe atentivamente la escena.
6. Experimentos con simulación y robot real hasta que funcionó correctamente.

A grandes rasgos este proyecto fin de carrera ha supuesto un paso adelante significativo sobre el proyecto [Martínez, 2005] extendiendo a 3D el algoritmo atentivo, incluyendo segmentos suelo además de balizas puntuales, combinado con la navegación del robot.

Los resultados de este proyecto, el código fuente de los programas desarrollados y vídeos de demostración, están disponibles en la web[1].

[1] http://gsyc.escet.urjc.es/jmplaza/research.html

6.2. Líneas Futuras

Uno de los caminos por los que se podría continuar este trabajo sería con una reconstrucción 3D más densa. Por ejemplo, en este proyecto se ha reconocido la posición 3D de balizas, pero no se ha hecho una estimación de su tamaño real.

Tampoco se ha intentado reconocer obstáculos que pudiera haber en el suelo, con los que potencialmente se puede chocar y podrían ser nuevos puntos de atención en el sistema.

Un paso más en esa misma línea sería la de utilizar el par estéreo de cámaras para reconstruir la escena de modo completo. Para ello habría que ir reconociendo cada uno de los puntos de las imágenes y realizando la pertinente proyección 3D. Así se podría construir una imagen de escena virtual mucho más densa.

Otro de los *objetos de interés* de la escena podrían ser personas. En este caso se podría hacer un seguimiento de atención sobre ellas, pero además de filtrando las imágenes por color, hacerlo por movimiento. De esta manera, cuando alguien se moviese, llamaría la atención del sistema. En esta dirección se está trabajando actualmente en el grupo [Palacios, 2006].

Se podría seguir en esa línea creando un algoritmo de navegación que siguiese a las personas que llamasen la atención sobre otras, por ciertas características. Por ejemplo, el algoritmo de atención posaría su mirada sobre las personas que tiene a su alrededor, pero sólo seguiría a la más alta. Y así hasta que encontrase otra más alta que la primera, etc.

Una posibilidad del algoritmo de atención por donde se podría seguir investigando sería la de que ésta estuviera guiada por hipótesis perceptivas. Por ejemplo, si el sistema hubiera detectado tres esquinas de un póster muy grande, se introduciría un punto de atención en la cuarta esquina no vista, para contrastar la hipótesis de que esos tres puntos corresponden a un rectángulo. De este modo se podrían descubrir determinadas formas en la escena más rápidamente.

Bibliografía

[Cañas, 2003] Cañas, J. M.: *"Jerarquía Dinámica de Esquemas para la generación de comportamiento autónomo"*, Tesis Doctoral, Universidad Politécnica de Madrid, 2003.

[Cañas, 2004] Cañas, J.M., Matellán, V., Montúfar, R.: *"Programación de robots móviles"*, Artículo para RIAI, Revista Igeroamericana de Automática e Informática Industrial, 2004.

[Cañas, 2005] Cañas, J.M., Martínez de la Casa Puebla, M.: *"Overt visual attention inside JDE control architecture"*, Artículo para EPIA, 2005.

[Cañas, 2004] Cañas, J.M.: *"Manual de programación de robots con JDE."*, URJC, 2004.

[Cañas, 2005] Cañas Plaza, J. M.: *Reconstrucción 3D mediante el algoritmo evolutivo de las moscas."*, Universidad Rey Juan Carlos, 2005

[Barrera, 2005] Barrera, P., Cañas Plaza, J. M., Matellán, V.: *"Visual object tracking in 3D with color based particle filter."*, Artículo para IJIT, 2005

[Bustos, 2003] Bustos, P.: *"Murphy: hacia un robot con visión estereoscópica"*, Tesis Doctoral, Universidad de Extremadura, 2003.

[Hidalgo, 2006] Hidalgo Blázquez, V.: *"Comportamiento de persecución de un congénere con el robot Pioneer"*, Proyecto Fin de Carrera, Universidad Rey Juan Carlos, 2006.

[Pineda, 2006] Pineda, A.: *"Localización 3D"*, Proyecto Fin de Carrera, Universidad Rey Juan Carlos, 2006.

[Peña, 2004] Peña Gutiérrez, S.: *"Podabot:Un robot podador"*, Proyecto Fin de Carrera, Universidad Rey Juan Carlos, 2004.

[Palacios, 2006] Palacios, R.: *"Escena rica"*, Proyecto Fin de Carrera, Universidad Rey Juan Carlos, 2006.

[Diaz, 2005] Díaz, P.M.: *"Navegación visual del robot pioneer"*, Proyecto Fin de Carrera, Universidad Rey Juan Carlos, 2005.

[Ortiz, 2004] Ortiz Herencia, R.: *"Comportamiento sigue persona con visión."*, Proyecto Fin de Carrera, Universidad Rey Juan Carlos, 2004.

[Calvo, 2004] Calvo Palomino, R.: *"Comportamiento sigue persona con visión direccional."*, Proyecto Fin de Carrera, Universidad Rey Juan Carlos, 2004.

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

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

[Martínez, 2003] Martínez de la Casa Puebla, M.: *"Comportamiento sigue pelota con visión cenital."*, Proyecto Fin de Carrera, Universidad Rey Juan Carlos, 2003.

[Martínez, 2005] Martínez de la Casa Puebla, M.: *"Sistema de atención visual en escena."*, Proyecto Fin de Carrera, Universidad Rey Juan Carlos, 2005.

[López, 2005] López Fernández, A.: *"Localización visual del robot Pioneer."*, Proyecto Fin de Carrera, Universidad Rey Juan Carlos, 2005.

[Isado, 2005] Isado, J. R.: *"Navegación global utilizando método del gradiente."*, Proyecto Fin de Carrera, Universidad Rey Juan Carlos, 2005.

[Lopez R, 2005] López Romero, A.: *"Navegación global utilizando grafo de visibilidad."*, Proyecto Fin de Carrera, Universidad Rey Juan Carlos, 2005.

[Pantilt, 2003] *"Computer Controlled Pan-Tilt Unit."*, User's Manual, 2003

[Borenstein, 1989] J. Borenstein.: *"Real-time obstacle avoidance for fast mobile robots.."*,IEEE Journal of Robotics and Automation, 1989.

[Hartley, 2004] Hartley, Richard and Zisserman, Andrew: *"Multiple View Geometry in Computer Vision"*.