UNIVERSIDAD REY JUAN CARLOS

# INGENIERÍA TÉCNICA EN INFORMÁTICA DE SISTEMAS

Escuela Superior de Ciencias Experimentales y Tecnología

Curso académico 2005-2006

**Proyecto Fin de Carrera**

# Comportamiento de persecución de un congénere con el robot Pioneer.

**Tutor**: José María Cañas Plaza

**Autor**: Víctor Manuel Hidalgo Blázquez

*Están los que usan siempre la misma ropa.*
*Los que llevan amuletos.*
*Los que hacen promesas.*
*Los que imploran mirando al cielo.*
*Los que creen en superticiones....*

*Y están los que siguen corriendo cuando*
*le tiemblan las piernas.*
*Los que siguen jugando cuando se les*
*acaba el aire.*
*Los que siguen luchando cuando todo*
*parece perdido, como si cada vez fuera*
*la última vez. Convencidos que la vida misma*
*es un desafio.*
*Sufren, pero no se quejan porque saben que*
*el dolor pasa, el sudor se seca, el cansancio*
*termina, pero hay algo que nunca desaparecerá:*
*La satisfacción de haberlo logrado.*

*En sus cuerpos hay la misma cantidad de músculos.*
*En sus venas corre la misma sangre.*
*Lo que los hace diferentes es su espiritu.*
*La determinación de alcanzar la cima.*
*Una cima a la que no se llega superando a*
*los demás, sino superandose a uno mismo.*

# Agradecimientos.

Quiero dar las gracias al grupo de robótica de la URJC. Destacando en especial a Jose María Cañas por los conocimientos facilitados, su paciencia, esfuerzo y apoyo.

Dentro del grupo de robótica, agradecer también a mis compañeros Antonio Pineda y David Méndez, por la ayuda prestada y por todo lo que hemos luchado juntos para poder realizar nuestros correspondientes proyectos.

A mi familia por aguantarme día a día y darme la posibilidad de estudiar.

A mis amigos por esos maravillosos ratos juntos que me hacen olvidarme de la realidad.

# Resumen.

Cada vez se venden más robots que entran en los hogares, como la aspiradora Roomba o el juguete Aibo. Un comportamiento potencialmente útil en ese escenario es el de seguimiento . En el grupo de *robótica* de la *URJC* se ha conseguido que un robot siga a una pared, una pelota y una persona. El objetivo de este proyecto es dar un paso más en esa línea y lograr que un robot *Pioneer* busque, identifique y persiga a otro robot *Pioneer*. Para ello deberá ser capaz de evitar obstáculos e identificar al congénere, mediante visión y el sensor láser. La dificultad de este proyecto reside en realizar una identificación robusta, para que no podamos engañarlo con objetos de color y tamaño parecidos, y lo detecte en casi cualquier orientación.

En la búsqueda se evitan los obstáculos tanto estáticos como dinámicos que se presenten, usando la técnica de navegación local *Campo de fuerzas virtuales (VFF)*. Con ella robot se acerca al objetivo, que tomaremos basándonos en las medidas del sensor láser y que se corresponde con un punto aleatorio del centro del pasillo. En la identificación en vez de aplicar técnicas de reconocimiento de patrones clásico de objetos, se ha diseñado un algoritmo basado en el principio etológico de *Suma heterogénea de subestímulos*. Así cuando la combinación de ciertos subestímulos (base roja, resalta en profundidad.....) supera cierto umbral se desencadena el comportamiento de seguimiento. En el seguimiento se persigue al congénere y se evitan obstáculos con la misma técnica usada en la parte de búsqueda, pero con la diferencia de que el objetivo esta vez es el robot detectado.

La forma de programar en C la funcionalidad que se busca ha sido a través de la plataforma *jde.c*. El proyecto se ha implementado por medio de cinco hebras o esquemas concurrentes, siendo cuatro de éstas perceptivas y una motora. La aplicación se ha probado extensamente en el robot real, comprobándose que reacciona en tiempo real a obstáculos, detecta al robot congénere y no se le engaña. Además se ha conseguido una gran flexibilidad en el comportamiento, ya que una vez que se está realizando el comportamiento de seguimiento, si se pierde al robot de vista se vuelve a buscarle.

# Índice general

# Índice de figuras

# Capítulo 1
# Introducción

Este proyecto fin de carrera, consiste en programar el robot *Pioneer* para que busque a otro robot, lo identifique y le persiga autónomamente, todo ello sin que se choque con ningún obstáculo.

En este capítulo se introduce el contexto en el que se desarrolla este proyecto fin de carrera: comienza con una reseña histórica de los inicios de la róbotica, repasando sus orígenes industriales y recorriendo algunos ejemplos de sus aplicaciones en la actualidad. A continuación se introduce un pequeño resumen a los términos *navegación y visión artificial* , dos campos en los que se sitúa este proyecto, y terminaremos con una breve descripción del proyecto fin de carrera realizado.

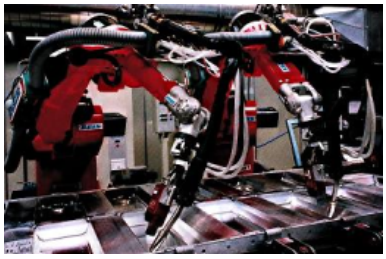
## 1.1. Robótica

Desde la revolución industrial el hombre ha tratado de satisfacer sus necesidades y liberarse de esfuerzo a través de la ciencia y más concretamente de robots. Robot es una palabra que proviene de la palabra *robota*, esta palabra apreció en 1921 en una obra del autor checo *Karel Capek* y donde llamaba de esta forma a unas máquinas que se ocupaban de todos los trabajos pesados para el hombre. En el siglo XIII aparecieron los primeros autómatas (realizaban tareas de forma mecánica).

La róbotica estudia los sistemas que realizan una conexión inteligente entre el sistema perceptivo y el de actuación. Así los robots están formados por sensores, que nos dan información del entorno en que se encuentra el robot, actuadores, a través de los cuales el robot interacciona con el mundo, y procesadores donde se coordinará la percepción y la actuación del robot.

A grandes rasgos existen dos tipos de robots. Están los *brazos manipuladores*, encontrados en la industria, y los *robots móviles* que nacen de la necesidad de resolver tareas más peligrosas y de investigación. En la industria, los más usados son los robots articulados. Estos robots son similares a los brazos humanos aunque poseen mucha mayor rapidez, precisión y potencia. Hoy en día podemos encontrar robot soldadores (figura 1.1 b), brazos cargadores(figura 1.1 a), sobretodo en el sector automotriz.

(a) Robot cargador



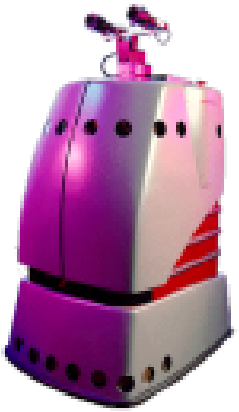
(b) Robot soldador

Figura 1.1: Robots industriales

Dentro de la robótica móvil podemos diferenciar entre: robots teleoperados y robots autónomos. Los primeros, cuyo movimiento es comandado a distancia por un operador humano, son usados para realizar tareas peligrosas o para realizar misiones en lugares donde el hombre tiene difícil accesibilidad , como la exploración interplanetaria. Ejemplos de estos robots son los robots *Opportunity* (figura 1.2(a)) y *Spirit* , aunque se les puede considerar parcialmente autónomos ya que, aunque se les indique qué tienen que hacer, son capaces de realizar autónomamente esas indicaciones.

Los robots móviles autónomos se caracterizan por decidir el movimiento a realizar según sea la información dada por sus sensores del entorno que los rodea. A día de hoy, estos robots son prototipos de investigación, se suelen usar en entornos cerrados aunque poco a poco se van introduciendo en espacios al aire libre. Un ejemplo de estos robots es *Minerva* (figura 1.2(b)), que actúa como guía en museos. Este proyecto fin de carrera, es otro claro ejemplo de autonomía, ya que basandose en la información proporcionada por los sensores el robot seguirá al otro robot o le buscará.

(a) Opportunity



(b) Minerva

Figura 1.2: Robots móviles

Los robots móviles autónomos se caracterizan por decidir el movimiento a realizar según sea la información dada por sus sensores del entorno que los rodea. A día de hoy, estos robots son prototipos de investigación, se suelen usar en entornos cerrados aunque poco a poco se van introduciendo en espacios al aire libre. Un ejemplo de estos robots es *Minerva* (figura 1.2(b)), que actúa como guía en museos. Este proyecto fin de carrera, es otro claro ejemplo de autonomía, ya que basandose en la información proporcionada por los sensores el robot seguirá al otro robot o le buscará.

La sociedad no solo utiliza los robots en el campo de la industria, también les da uso en otros campos como ocio, educación, hogares, medicina etc. En el sector de entretenimiento, son usados de mascotas y como jugadores de fútbol en la exhibición mundial de la *Robocup* [1] (figura 1.3(a)). En los últimos años se están realizando grandes avances en los modelos humanoides, como el *QRIO* de Sony y *ASIMO* (figura 1.3(b)de Honda , consiguiendo comportamientos tan complejos como andar , subir , bajar escaleras y correr.En el sector de la educación podemos encontrar el *Lego Mindstorm* como kit de construcción y programación con piezas *Lego*. En la medicina se han introducido brazos robot izados que bajo el mando de un cirujano, intervienen quirúrgicamente a pacientes.

[1] http://www.robocup.org

(a)

(b)

(c)

Figura 1.3: exhibición robocup (a) , ASIMO (b) y Roomba (c)

Un último ejemplo del gran uso de la robótica en la sociedad es el robot *Roomba* (figura 1.3(c)), el cual aspira y barre suelos, y al que podemos considerarle como un *best seller* de la róbotica ya que se han vendido más de un millón de ejemplares.

## 1.2. Navegación de robots

El primer problema de un robot es cómo moverse por su entorno. Así 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, bien buscando al otro robot o bien persiguiéndole.

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

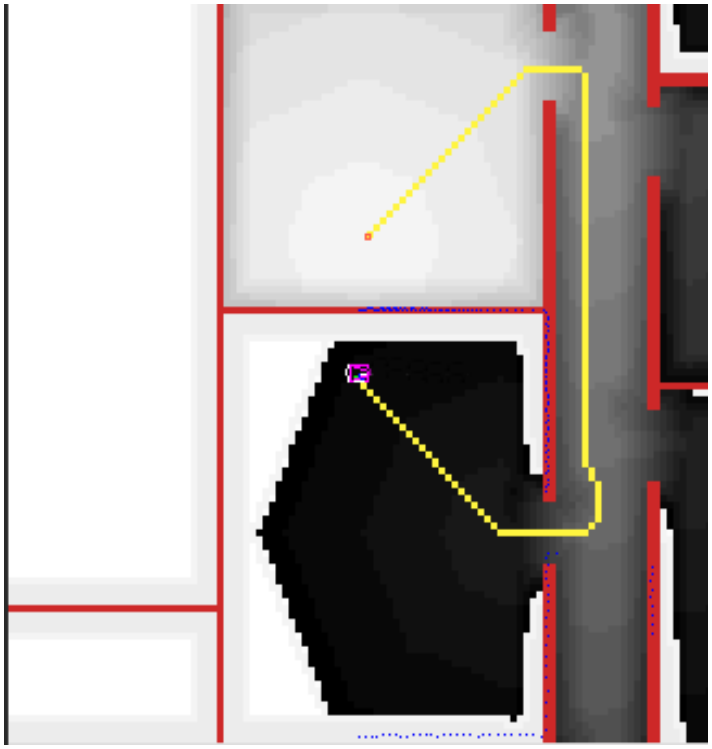


Figura 1.4: Navegación global basada en gradiente

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, cuyas técnicas son bastantes costosas. 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 lo 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) , *Diagrámas de voronoi* y *planificación basada en gradiente*(Konolige, 2000). En la figura 1.4 podemos ver un ejemplo de navegación global [Isado04] 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, navegar 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. Una gran ventaja frente a la *navegación global* es la evitación de osbáculos dinámicos, útil para este proyecto.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 aproximan a círculos. Así el objetivo para este método 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(VLM): Este método combina el *CVM* con un nuevo método llamado el método de carriles. El método de carriles divide el entorno en carriles, y luego elige el mejor a seguir hacia la dirección deseada. Así se consigue un movimiento libre de colisiones como un movimiento suave teniendo en cuenta la dinámica del robot.

- Campos de fuerza 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. Esta fuerza es mayor, cuanto más cerca está el robot del objetivo. Sin embargo no es esta 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. Así el movimiento del robot se ve gobernado por la suma vectorial de todas la fuerzas que afectan al robot. Esta técnica tiene el compromiso de alejarse de los obstáculos y llegar al objetivo deseado.

Debido a que estos dos tipos de navegación tienen inconvenientes, surge la *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*[2] y el vehículos *Stanley* ganador de la carrera *Darpa Grand Challange* [3], que se realiza cada año y donde vehículos no tripulados han de realizar un recorrido.

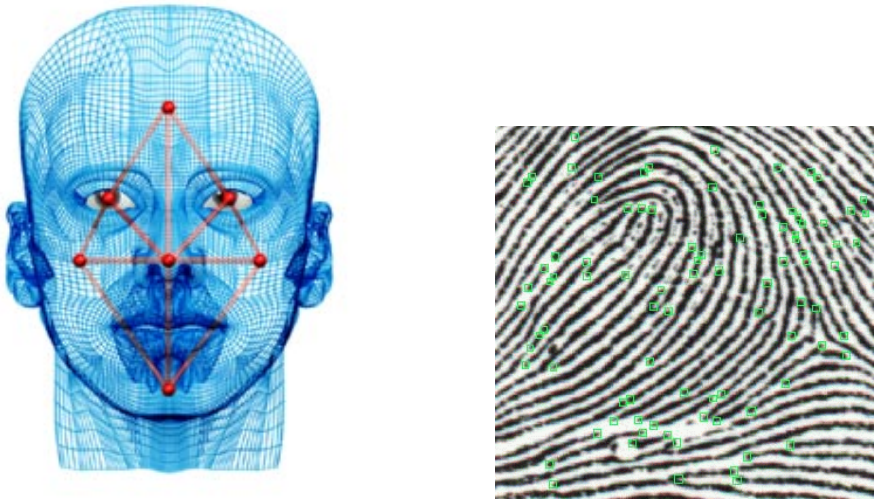
## 1.3. Visión Computacional

La *visión computacional* ( el sentido de la vista de la computadora ) es una disciplina que intenta emular la capacidad que tienen algunos seres vivos para extraer información útil desde las imágenes.La visión computacional ha experimentado una

[2]http://www.es.emu.edu/Xavier
[3]http://www.grandchallenge.org

rápida evolución en las últimas dos décadas. Entre las razones de este avance se puede destacar el incremento de prestaciones del hardware (se realizan cálcuos en un tiempo razonable), el uso dado en áreas como la *robótica* y gráficos por computador y, al crecimiento de aplicaciones industriales que demandan el uso de estas técnicas.

Este tremendo desarrollo tecnológico ha sido decisivo para la consolidación de áreas muy lucrativas como los videojuegos o aplicaciones gráficas avanzadas tipo simuladores de vuelo, reconocimiento de matrículas de vehículos, reconocimiento facial (figura 1.5(a)), reconocimiento de iris, reconocimiento de huellas dactilares (figura 1.5(b) , etc.



(a) Reconocimiento facial

(b) Reconocimiento de huellas dactilares

Figura 1.5: Reconocimiento facial y de huellas dactilares

Un problema fundamental en visión computacional es el reconocimiento de objetos , es decir, cómo realizar el etiquetado de estos estímulos tan complejos asignándoles un nombre. De echo en este proyecto tendremos este problema para poder reconocer al congénere, del otro robot.Tradicionalmente para poder identificar un objeto , se usa el *reconocimiento de patrones clásico*, dividido normalmente en cuatro fases:

1. La primera fase consiste en la captura o adquisición de las imágenes.

2. La segunda etapa consiste en el tratamiento digital de las imágenes, con objeto de facilitar las etapas posteriores. En esta etapa es donde se realizan filtros ,

segmentaciones , transformaciones geométricas o se eliminan partes indeseables de la imagen

3. En la tercera fase se seleccionan y se extraen las características del objeto

4. Y por último usando algún clasificador se llega o no a la identificación de cierto objeto. Algunos ejemplos de clasificadores son los clasificadores de características discriminantes (ver figura 1.6) que convierten un objeto en un vector y usando una función determinante identifican el objeto, clasificadores de patrones que se pueden ordenar atendiendo a diferentes criterios(a la forma de construirse, el tipo de muestra, información disponible..) y los clasificadores basados en las distancias, los cuales usan métodos estadísticos.

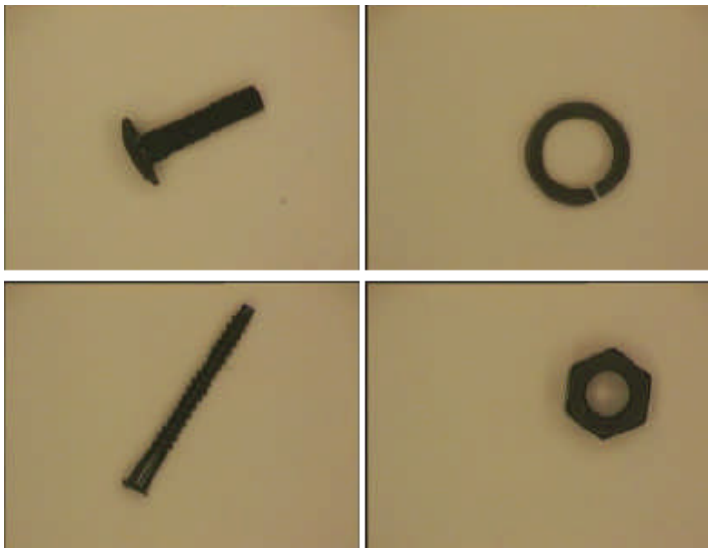


Figura 1.6: Clasificador de características discriminantes

Hoy en día , se está empezando a utilizar cada vez más la visión debido a que las cámaras son muy baratas. Desde el punto de vista de la robótica las cámaras son un sensor más que proporcionan mucha información del entorno. Las cámaras entregan un flujo continuo de imágenes,pero es difícil extraer de ese flujo información útil para la acción del robot. La extracción de información desde las imágenes es muy compleja y costosa, por estos motivos hay que llegar al compromiso de sacar información sin perder tiempo real. Así la visión al proporcionar información del entorno puede ser utilizada para generar un comportamiento en el robot. En esta línea aparecen robots capaces de seguir objetos en movimiento(para ello ha tenido que identificarlo) o navegar por un entorno. Un ejemplo del gran uso que la robótica le esta dando a la visión computacional, es el hecho de que casi todos los robots modernos tengan cámaras (ojos), como

puedan ser los perritos (figura *Aibo*1.3(a)) o el robot *Minerva* (figura 1.2(b).

## 1.4. Comportamiento de persecución del robot Pioneer

Una vez presentado el contexto general de la robótica y dos de los campos en los que se sitúa este proyecto ( navegación y visión computacional) vamos a describir el contexto más cercano que motivo este proyecto, el cual está enfocado a generar un comportamiento autónomo de seguimiento sobre un robot real.

Actualmente en la sociedad, existe cada vez más robots que entran en los hogares para desempeñar una función de ayuda al hombre. Esto implica que el robot deba de realizar seguimientos . Por eso el generar comportamientos autónomos de seguimiento se muestra como una gran utilidad. Asi siguiendo la línea de investigación del grupo, este proyecto junto con los proyectos del sigue persona ( [Calvo, 2004]), sigue pared ([Gomez, 2002]) , sigue pelota ([Martinez, 2003]) tratan de resolver los comportamientos de seguimiento en los robots.

El comportamiento consiste en que el robot busque a otro robot igual en entornos de interiores , sin chocarse, y le persiga. Para conseguir que no se choque hemos usado una técnica de navegación local. La parte más novedosa en este proyecte es la de identificación del robot.

Para intentar conseguir una identificación más robusta que el reconocimiento de patrones nos hemos basado en el *principio de suma heterogénea de estímulos*. El *principio de suma heterogénea de estímulos* , [*Konrad Loref,1978*], dice que si la suma de un conjunto de estímulos supera cierto umbral motivan los comportamientos y/o producen la desencadenación de comportamiento. Un ejemplo de este principio es la apertura del pico de las crías de gaviota cuando viene el progenitor para alimentarles , intervienen muchos factores: la forma del pico, una manchar roja en el pico ,forma y color de la cabeza etc.

Para tratar de explicar cómo se llevó a cabo, esta memoria está organizada en primer lugar con un capítulo dedicado a los objetivos y requisitos que ha de cumplir el comportamiento. Posteriormente, en el capítulo de herramientas, se analiza con detalle el robot sobre el que se realizaron los experimentos, y los diferentes dispositivos de los que dispone. En la parte de software hablaremos de la plataforma de programación sobre la que se ha desarrollado el comportamiento llamada *JDE*. Seguidamente encontraremos un capítulo dedicado a la descripción informática de este comportamiento, donde se explica la solución desarrollada para este proyecto, describiendo los programas diseñados para conseguir el comportamiento de seguimiento.Una vez comentada la solución analizaremos los experimentos realizados. Finalmente se describirán las conclusiones y una línea de posibles trabajos futuros sobre el mismo proyecto.

# Capítulo 2
# Objetivos

Una vez expuesto el contexto de este proyecto en el capítulo de introducción , vamos a fijar los objetivos y requisitos concretos de este proyecto fin de carrera que han condicionado su desarrollo.

El objetivo principal es programar el robot *Pioneer* para que persiga a otro robot móvil, para ello el robot deberá ser capaz de deambular autónomamente por el entorno sin chocar contra ningún obstáculo, visualizar al otro robot cuando aparezca y por u'ltimo perseguirle.

## 2.1. Descripción del problema

Este proyecto tiene como objetivo dotar al robot de la capacidad de búsqueda de su *congénere*, una vez encontrado se seguirá a dicho robot. Además el robot ha de ser capaz de moverse sin chocar con ningún obstáculo , tanto dinámico como estático. Los subobjetivos en los que se divide este comportamiento son :

1. Desarrollar un algoritmo para que el robot deambule buscando al otro robot *Pioneer*. Para ello deberá navegar por el entorno sin chocarse con ningún obstáculo.
2. Diseñar y Desarrollar un método de identificación del robot. Esta identificación a de ser más robusta que el *reconocimiento de patrones cláscico*.
3. Desarrollar la persecución del robot, al igual que en el punto uno el robot deberá navegar por el entorno sin chocarse y a la vez deberá de ser capaz de seguir al otro robot *Pioneer*.

El punto de partida para el desarrollo de este proyecto es el robot Pioneer dotado con sensores odométricos, de visión (cámara), y el láser, todos ellos nos darán infor-

mación del entorno en el que se encuentra el robot. Este proyecto esta pensado para entornos de interiores como puedan ser los pasillos del departamental de la *ESCET*.

## 2.2. Requisitos

El desarrollo de este proyecto está guiado por los objetivos comentados anteriormente y deberá ajustarse a los requisitos de partida para asegurar un buen comportamiento :

1. El comportamiento debe de funcionar sobre el robot *Pioneer 3-DX*, pudiéndose apoyar en herramientas de simulación como es *Player/Stage*, sin embargo el proyecto final debará funcionar en el robot real.Como fuente de datos nos hemos basado en dos sensores. Por un lado el sensor láser, que nos proporcionara información del entorno para la navegación, y por otro la cámara de la cual obtendremos información para la identificación del robot *Pioneer*.

2. Para la generación del comportamiento deseado debe usarse la plataforma software *JDE*, que será descrita en el siguiente cápitulo, y bajo un sistema operativo Linux. En concreto debe funcionar sobre las versiones *JDE (3.4) , oculo (3.8) y otos (5.2)*. Este entorno establece la programación de la aplicación con el lenguaje *C*.

3. La navegación debe de ser reactiva en todo momento y vivaz. Los algoritmos de navegación no pueden perder mucho tiempo pensando el próximo moviento ha realizar, porque ha de reaccionar rápido para evitar la colisión con cualquier posible obstáculo.

4. El algoritmo de identificación a de ser muy robusto, para que el robot no sea engañado con cualquier otro objeto de color y tamaño parecido, como pueda ser un poster en la pared del robot *Pioneer*. Además se tiene que poder identificar al robot en condiciones de luminosidad variables.

## 2.3. Metodología

La metodología utilizada para la realización de este proyecto, básicamente ha consistido en realizar varias iteraciones dentro de cada subobjetivo que se componen de: diseño, implementación y experimentos.

El desarrollo de este proyecto se ha basado en el modelo de desarrollo en espiral basado en prototipos. La elección de este modelo de desarrollo se basa en la necesidad de separar el comportamiento final en varias subobjetivos más sencillas para luego fusionarlas. Estos subobjetivos son independientes entre si, una vez terminados todos los subobjetivos habrá una fase final de pruebas de todo el sistema integrado.

La ventaja principal del modelo en espiral es que, al basarse en prototipos, en cada iteración existen puntos de control que ayudan a afrontar el proyecto mediante etapas. Además es muy flexible , en cuanto al cambio de requisitos, muy común en este tipo de proyectos de investigación.

Como podemos observar en la figura 2.1, el modelo de desarrollo consta de cuatro etapas : **Análisis de requisitos, diseño e implementación, pruebas y planificación del próximo ciclo de desarrollo**.

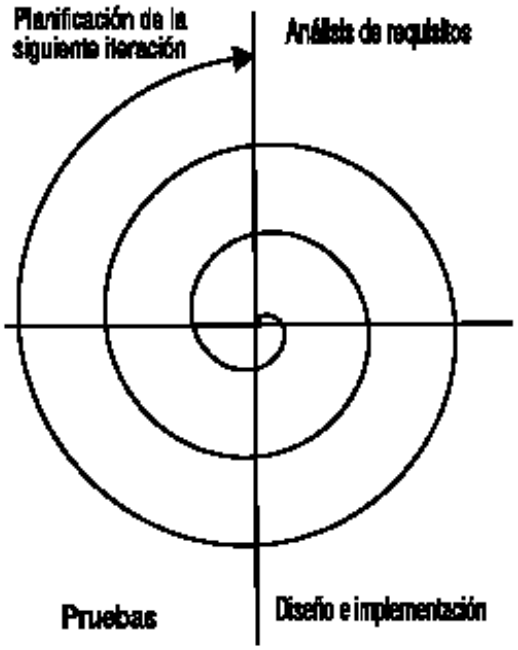



Figura 2.1: espiral

A lo largo de este proyecto se han implantado reuniones semanales con el tutor de este proyecto para establecer los puntos a llevar a cabo en cada etapa del mismo y comentar los resultados de etapas anteriores.

Las iteraciones generados a lo largo de estos diferentes subobjetivos han sido:

1. Introdución a la plataforma software:
   - *Prototipo 1.* Familiarización con la plataforma software **JDE**, la cual describiremos en el capítulo 3, utilizada en la implementación de este proyecto.(Un meses).
2. Navegación:
   - *Prototipo 2.* Generación de un algoritmo de navegación local.(Dos meses)
   - *Prototipo 3.* Generar el comportamiento de deambular (una semana).
   - *Prototipo 4.* Navegación local + deambular(una semana).
3. Identificación del robot.
   - *Prototipo 5.* Identificación del robot(cinco semanas).
4. Persecución
   - *Prototipo 6.* Seguimiento del robot(dos semanas).
5. Integración de subobjetivos
   - *Prototipo 7.* Experimentos globales con todos los subobjetivos integrados. (una semana).

# Capítulo 3
# Plataforma de desarrollo

En este capítulo presentamos las herramientas donde nos hemos apoyado para realizar el proyecto, el robot donde se ejecuta la aplicación , así como la plataforma software sobre la que se ha realizado.Por último comentaremos algunas bibliotecas y simuladores , en los que nos hemos apoyado.

## 3.1. Plataforma hardware: Robot Pioneer

El robot en el que se desarrolla la aplicación es el **Pioneer 3DXE**, y está comercializado por la empresa *ActivMedia*[1]. El *Pioneer* está dotado de unos sensores que le permiten medir el entorno, unos procesadores que le dan capacidad de cálculo y unos actuadores que le permiten moverse.

En el apartado de procesamiento el robot dispone de un único procesador de 18 MHz Hitachi H8S/2357 con 32Kb RAM y 128Kb Memoria flash. Tiene una autonomía de 5 horas , es capaz de adquirir una velocidad lineal de 1'8 *m/s* y una velocidad angular de 360 *grados/s*.Puede llegar a soportar una carga de 23 Kg.Encima de toda esta base física, hemos colocado un ordenador portátil, donde ejecutaremos nuestra aplicación

En el apartado sensorial dispone de unos odómetros(*encoders*)que cuentan las vueltas de cada rueda(tres ruedas). Gracias a estos odómetros obtendremos información de posición (v,w,$\Theta$).También posee una corona de 16 sensores de ultrasonido. A este equipamiento hemos añadido un láser *SICK*[2] y dos cámaras firewire *iSight*[3].Los actuadores principales son dos motores de continua, cada uno asociado a una rueda motriz, con ellos se dota al robot de un movimiento de tracción diferencial (tipo tanque).

[1] http://www.activrobots.com/ROBOTS/index.html#p2dx
[2] http://www.sick.es/es/productos/autoident/medicion_laser/interior/es.html
[3] http://www.apple.com/es/isight/

(a) Robot Pioneer

(b) cámara iSight

Figura 3.1: Robot pioneer y cámara iSight

## 3.2. Plataforma software

El software se basa en la plataforma **JDE** (*jerarquía dinámica de esquemas*) [Plaza, 2003], y mejorada por el *grupo de Robótica* de la *URJC*[4].

*JDE* propone una organización de la aplicación robotica basada en *esquemas*. La programación con *esquemas JDE* define la aplicación como un conjunto de esquemas que se ejecutan simultáneamente y en paralelo. Los esquemas no son más que procesos que funcionan concurrentemente y se encargan de una tarea sencilla y concreta. La ejecución simultánea de varios esquemas dan lugar a un comportamiento. Existen dos tipos de **esquemas: perceptivos y de actuación**. Los *esquemas perceptivos* se encargan de producir y almacenar información que puede ser leída por otros esquemas. Esta información puede provenir de medidas sensoriales o de la información elaborada por otros esquemas. Los *esquemas de actuación* toman decisiones sobre motores o activación de esquemas de niveles inferiores a partir de la información que generan los esquemas perceptivos. Los *esquemas* pueden *organizarse en niveles* estableciendo una jerarquía entre esquema padre y esquema hijo siendo el padre el que puede activar o desactivar al hijo.

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

La comunicación , en esta jerarquía de esquemas ,con los sensores y los actuadores está facilitada por dos servidores: **Otos y Oculo**. Cada servidor encapsula ciertos sensores y actuadores que hay en su máqina y ofrece su funcionalidad al resto de los programas a través de un *API de varialbes*.

El servidor *Otos* , se encarga de dar información de los sensores de proximidad (sónar , láser y táctiles), también genera y ofrece la posición , velocidad lineal , angular del robot a través de los odómetros y permite enviar comandos de movimiento a la base. Este servidor para dar la información al programador de los sensores lo que hace es escribir en ciertas variables los datos obtenidos, así si el programador necesita saber la posición del robot le bastará con consultar las variables *robot[0] y robot[1]*, y si quiere que el robot valla a una determinada velocidad le bastará con escribir en la variable *v*.

El servidor *Oculo* se encarga del movimiento del cuello mecánico y de los sensores de imagen (cámaras). Asi si queremos obtener una imágen tendremos que leer la variable *colorA* que es actualizada por el servidor oculo.

Si la programación fuera directa sobre estos servidores , obliga a la aplicación a encargarse del envío y recepción de mensajes con los servidores, lo que llevaría a un cierto esfuerzo al programador. Sin embargo , la programación con esquemas libera de esa tarea al programador al incorporar varios *esquemas de servicio* que se encargan de la comunicación : recogen de los servidores información de los sensores y envían a los motores ordenes de movimiento a través de dichos servidores.

Entre los clientes ,que controlan el robot , y los servidores se establece una conexión *tcp* , a través de la cual se produce un dialogo entre ambos regido por un protocolo. Dicho *protocolo JDE* está pensado para el envío continuo de mensajes (*streaming*) y en él se han establecido tres patrones de interacción: *la suscripción a sensores cortos, envío bajo demanda de imágenes y las órdenes de movimiento.*Así la escritura o consulta a una variable que se utilizan en el acceso local se sustituyen en el acceso remoto por el envío de cierto mensaje . Si localmente las variables eran actualizadas por una tarea, ahora esas variables no residen en la máquina donde está el sensor y la tarea de actualización incluirá su petición al servidor correspondiente y la lectura desde la red de las respuestas oportunas. Del mismo modo la llamada a función con que se comando un movimiento se sustituye por el envío del mensaje correspondiente al servidor que realmente materializa esas órdenes a los motores ( ver figura 3.2 )

.

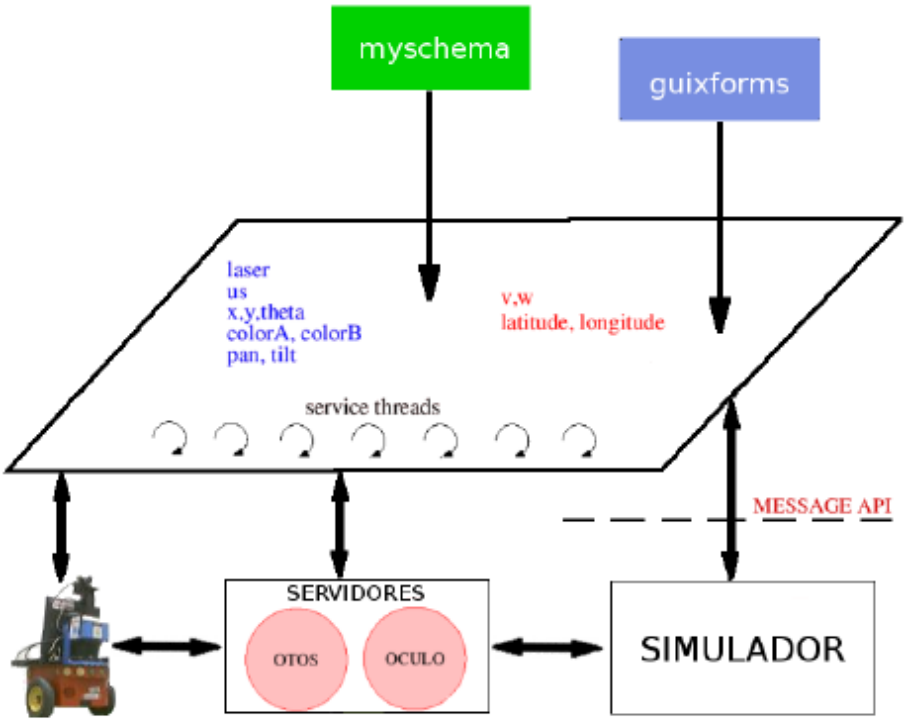



Figura 3.2: jde-básico

Esta arquitectura de servidores y clientes *JDE* , ha conseguido que un programa no tenga que ser ejecutado obligatoriamente en el portátil a bordo del robot.Permite aprovechar mejor la capacidad de cómputo disponible . Facilita el procesamiento distribuido y paralelo de la información que proviene de los sensores. Hace mas independiente a los programas del robot concreto.Otra característica ventajosa que heredan los servidores y clientes de *JDE* es que el servidor otos se puede conectar indistintamente a la *plataforma real o a un simulador*.

Sin embargo , estas ventajas dan lugar al *aumento de retardo* entre que se lee un dato sensorial y este llega , atravesando los servidores y red, al programa donde es procesado realmente.

## 3.3. Simulador *player stage*

Para realizar pruebas de *navegación* en nuestro proyecto antes de realizarlas en el robot real y evitarnos algún susto, hemos utilizado este simulador.

La plataforma **Player/Stage** [Brian P. Gerkey y Howard, 2003] es otro entorno de programación de aplicaciones robóticas. El entorno proporciona el servidor de dispositivos robóticos *Player*, que da acceso a sensores y actuadores desde programas clientes, y el simulador de múltiples robots *Stage*.Queremos además usar esta plataforma desde *JDE*.

*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*. Los clientes no notan ninguna diferencia entre los dispositivos reales y su equivalente simulado en *Stage*. El hecho de simular varios robots a la vez, nos va a permitir tener obstáculos dinámicos, que no aparecen en el mapa.

*Stage* puede además simular cualquier entorno, ya que se recibe este como una imagen en formato PNM. Esto supone que se puedan representar entornos irregulares o curvos( ver figura 3.3 ).

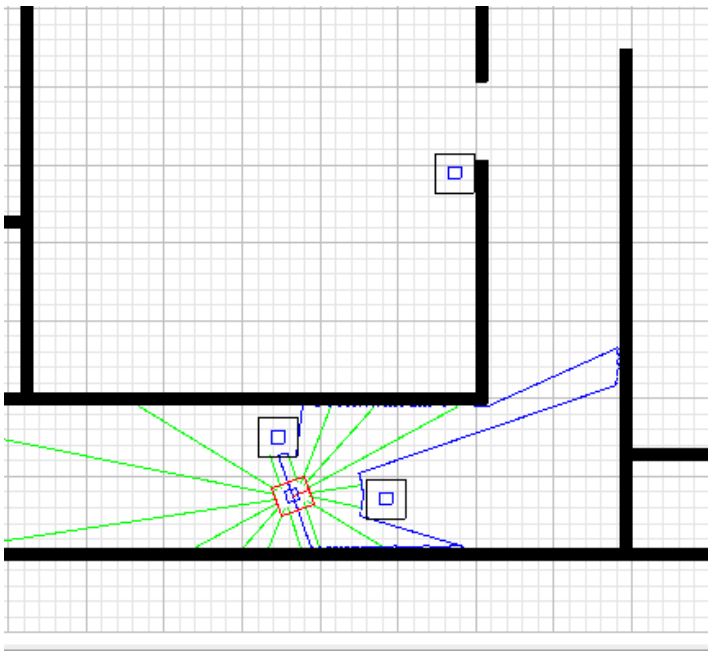


Figura 3.3: simulador Player/Stage

## 3.4. Bibliotecas auxiliares: *Fuzzylib* y *PROGEO*

Las bibliotecas Fuzzylib y PROGEO se han usado para proporcionar distintas funcionalidades a los esquemas programados.

La biblioteca *Fuzzylib* aporta funcionalidad para usar controladores borrosos. La lógica borrosa es básicamente una lógica multivaluada que permite valores intermedios más ala de las evaluaciones binarias como sí/no., verdadero/falso, negro/blanco. Las nociones como "más bien caliente.º "poco frío"pueden formularse matemáticamente y ser procesadas. Así podemos tener valores intermedios entre verdadero y falso, como "probable.º "muy poco probable".
El controlador borroso define etiquetes para las variables de entrada y salida que maneja. Como por ejemplo en la temperatura: muy frío, frío, templado, tibio, caliente, muy caliente.Dicho controlador hace uso de un fichero de reglas del tipo

```
IF THEN
```

, que ofrece un valor de salida según sea el valor de entrada.

En este proyecto usaremos un controlador borroso para controlar las velocidades lineales y angulares del robot, que lo pilotarán en su navegación a través de su entorno como explicamos en el capítulo 4.

La información que hay en una imagen es mucha. Sin embargo,la información se encuentra de manera explícita en la imagen. La biblioteca *PROGEO* (*PROjective GEOmetry*) nos ayuda a conseguir información tridimensional, obteniendo un punto en 3D a partir de un pixel y viceversa.

*PROGEO* esta **basado** en el modelo de cámara **pin-hole**. Así cualquier imagen captada por la cámara pasa a través del foco proyectándose en un plano y creando una imagen invertida respecto de la imagen real ( ver figura 3.4 ). Como podemos ver en la figura la distancia entre el plano de imagen y el foco se corresponde con la llamada *distancia focal*. Para no trabajar con la imagen invertida, *PROGEO* trata con un plano que se encuentra entre el foco y la imagen real (plano de proyección), dicho plano se encuentra a una distancia respecto del foco igual a la *distancia focal*. Además para poder extraer información correcta basándonos en el modelo *pin-hole*, tenemos que conocer los parámetros extrínsecos e intrínsecos de la cámara. Los parámetros inttrínsecos son,

distancia focal y el pixel por donde pasa el foco en el plano de proyección, para conseguir estos valores hemos tenido que calibrar la cámara usando para ello la biblioteca *OpenCV*. Los parámetros extrínsecos son aquellos que tiene que ver con la posición de la cámara , coordenadas (x,y,z) con respecto de un sistema de referencia (en nuestro caso el centro del robot) , ángulo de la cámara respecto la horizontal y las coordenadas 3D hacia donde apunta el foco.

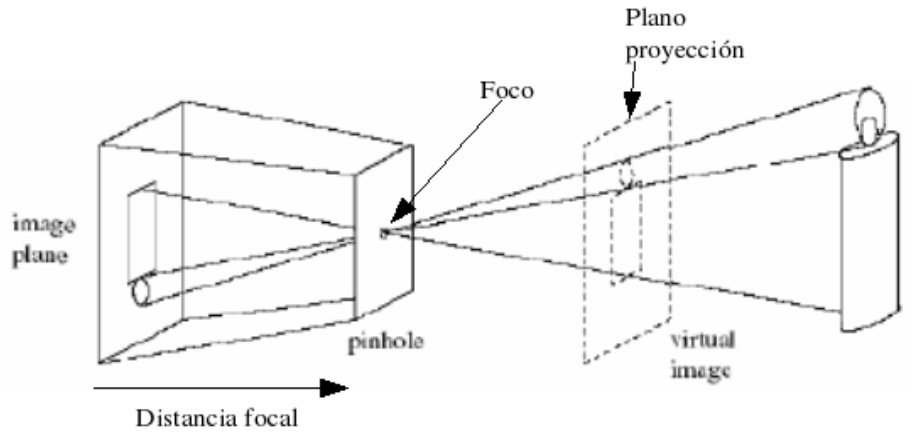



Figura 3.4: modelo pin-hole

*PROGEO* ofrece las siguientes funciones:

1. *project*, esta función recibe como entrada un punto en 3D del espacio y devuelve su correspondiente pixel en el plano de proyección. Para ello *PROGEO* traza una línea entre el punto 3D y el foco, y el punto de corte entre esta línea y el plano de proyección será el pixel que nos devolverá la función,

2. backproject, esta función realiza la operación inversa que la anterior, es decir, tomando como entrada un pixel nos devuelve su correspondiente punto en 3D del espacio. Para ello trazamos una linea entre el foco y el pixel, así obtendremos la linéa donde se encuentra el punto en el espacio pero para saber exactamente donde esta el punto necesitamos dos cámaras. Así cada cámara trazará su recta, y el punto donde se corten nos dará el punto correspondiente en 3D.

3. *update matrices*, esta función nos permite actualizar ciertas matrices , utilizadas para realizar las anteriores transformaciones (obtener un pixel a partir de un punto en 3D y viceversa), cuando se produce algún cambio en los parámetros intrínsecos o extrínsecos.

# Capítulo 4
# Descripción informática

En este capítulo describiremos la solución desarrollada para alcanzar el objetivo propuesto en el capítulo 2 e implementando ésta sobre la plataforma descrita en el capítulo anterior. Para ello veremos primero una visión global de cómo hemos estructurado el comportamiento en forma de esquemas *JDE* y luego explicaremos más detalladamente cada uno de los esquemas.

## 4.1. Diseño general

El objetivo de este proyecto es que un robot *Pioneer* busque a otro robot *Pioneer*, lo identifique y lo persiga. La búsqueda consiste en localizar al robot en un entorno cerrado, donde evitará los obstáculos tanto estáticos como dinámicos que se le presenten, utilizando para ello la técnica de navegación local: *Campo de fuerzas virtuales (VFF)* [Borenstein, 1989]. La identificación consiste en la detección del robot mediante visión y láser, para ello en lugar de aplicar técnicas de reconocimiento de patrones clásico de objetos, se ha diseñado una técnica basada en un principio etológico: *Suma hetereogénea de estímulos* [Lorenz, 1978], comentado en el capítulo 1. Una vez realizada la identificación, comienza la parte de seguimiento, en la que el robot perseguirá a su congénere y evitará obstáculos con la misma técnica usada en la primera parte, pero con la diferencia de que el objetivo esta vez es el robot.

El comportamineto buscado está implementado mediante cinco esquemas 4.2, como los que hemos explicado en la sección 3.2. La búsqueda está implementada con los esquemas perceptivos *mempuntos, lasersegments* y el esquema motor *vff*. La identificación, está realizada a través de los esquemas *imagesegments, identity y lasersegments*. El seguimiento se lleva a cabo mediante los esquemas perceptivos *mempuntos, identity* y el esquema motor *vff*. La diferencia entre la parte de búsqueda y seguimiento, está en el objetivo hacia donde tiene que ir el robot, que es diferente. Así si el congénere es

identificado el esquema *identity* nos da como objetivo para navegar la posición del robot identificado, si no es así el esquema *vff* basándose en el esquema *lasersegment* obtendrá como objetivo un punto aleatorio al final del pasillo, para navegar explorando el entorno, buscando al otro robot.

Estos cinco esquemas, con los que se lleva a cabo el comportamiento, están activos a la vez. Este hecho permite tener una gran flexibilidad en el comportamiento, ya que se sigue al robot cuando lo identifique, que es cuando tiene sentido, sin seguirle durante un tiempo predeterminado. Además cuando lo perdamos, volvemos a buscarle. De esta manera podemos ver este comportamiento como un autómata, figura 4.1, donde inicialmente el robot se encuentra en el estado de búsqueda. Al producirse la identificación del congénere se transita al estado de seguimiento. Una vez en este estado si no se localiza al robot pasaremos al estado inicial.

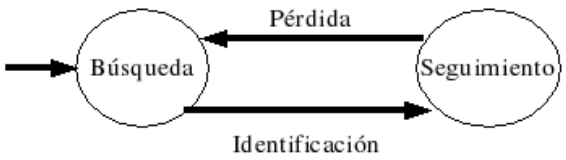



Figura 4.1: secuencia flexible del comportamiento

Para poder comandar al robot correctamente, evitando los obstáculos y seguir al otro robot cuando sea identificado, nos basamos en la información obtenida por los sensores. Por un lado mediante el sensor láser obtendremos información útil para navegar e identificar, por otro lado obtendremos información visual mediante la cámara, útil para identificar al otro robot. Así tenemos dos esquemas encargados de tomar los datos del entorno,*mempuntos e imagesegments*, por lo tanto son *esquemas perceptivos*. El esquema *mempuntos* tiene como finalidad crear un mapa local, para poder tener un mayor alcance y tener presente objetos como los que dejamos atrás, tomando como entrada los datos del láser. El esquema *imagesemgents*, se encarga de procesar las imágenes de la cámara ofreciendo a otros esquemas información más elaborada como las agrupaciones de color rojo, azul y negro en la imagen.

Una vez que tenemos almacenada la información del entorno en forma de segmentos de color y de un mapa local, tendremos diferentes esquemas que se encargarán de calcular un objetivo al que debe de dirigirse el robot. El esquema perceptivo *laser-*

*segments* basándose en el mapa local se encarga de segmentar el entorno, identificar el pasillo y su punto central. El esquema perceptivo *identity* tomando como entrada las agrupaciones de colores ofrecidas por el esquema *imagesegments* y apoyándose en la segmentación del entorno (*lasersegments*), extraerá las características de cada segmento y basándose en el *principio de suma heterogénea de estímulos* identificará al robot, o dará un posible candidato a ser el robot. Este esquema pondrá a disposición del esquema actuador la posición del objeto identificado como robot congénere, o un candidato si no se ha identificado a éste.

Una vez sabiendo dónde se encuentran los obstáculos, y si se ha detectado el robot o un candidato mediante los esquemas *mempuntos y identity* respectivamente,tendremos que comandar el robot dándole unas velocidades lineal y angular determinadas. De todo esto se encarga el esquema *vff*, esquema *actuador* que comanda el robot, donde está implementado el algoritmo de navegación local, este algoritmo usa la técnica del *vff* comentado en el capítulo 1. Si este esquema no identifica ningún robot y no existe ningún candidato, tomará como objetivo el final del pasillo que es dado por el esquema *lasersegments*.

El diseño de la aplicación sobre la plataforma *jde.c* se observa en la figura 4.2. Hay tres partes bien definidas. La primera parte son los esquemas perceptivos, dentro de rectángulos, recogen y procesan información de las variables sensoriales ( los datos del láser y la imagen, que proporciona *jde.c*, se encuentran en las variables *laser[] y colorA*), poniéndola a disposición de los demás esquemas. La segunda parte es el esquema actuador, dibujado en circulo, comanda las ordenes para generar el movimiento del robot (actualizando para ello las variables de actuación que pone a nuestra disposición *jde.c*). La última parte es la plataforma *jde.c* que actualiza las variables sensoriales y materializa las variables de actuación. Además los esquemas tendrán el color amarillo, si intervienen en la identificación, azul si forman parte de la navegación, y ambos colores si se utilizan en las dos partes. También podemos ver en verde las variables que actualizan iterativamente los esquemas implementados y que son utilizados por otros esquemas para generar el comportamiento deseado.

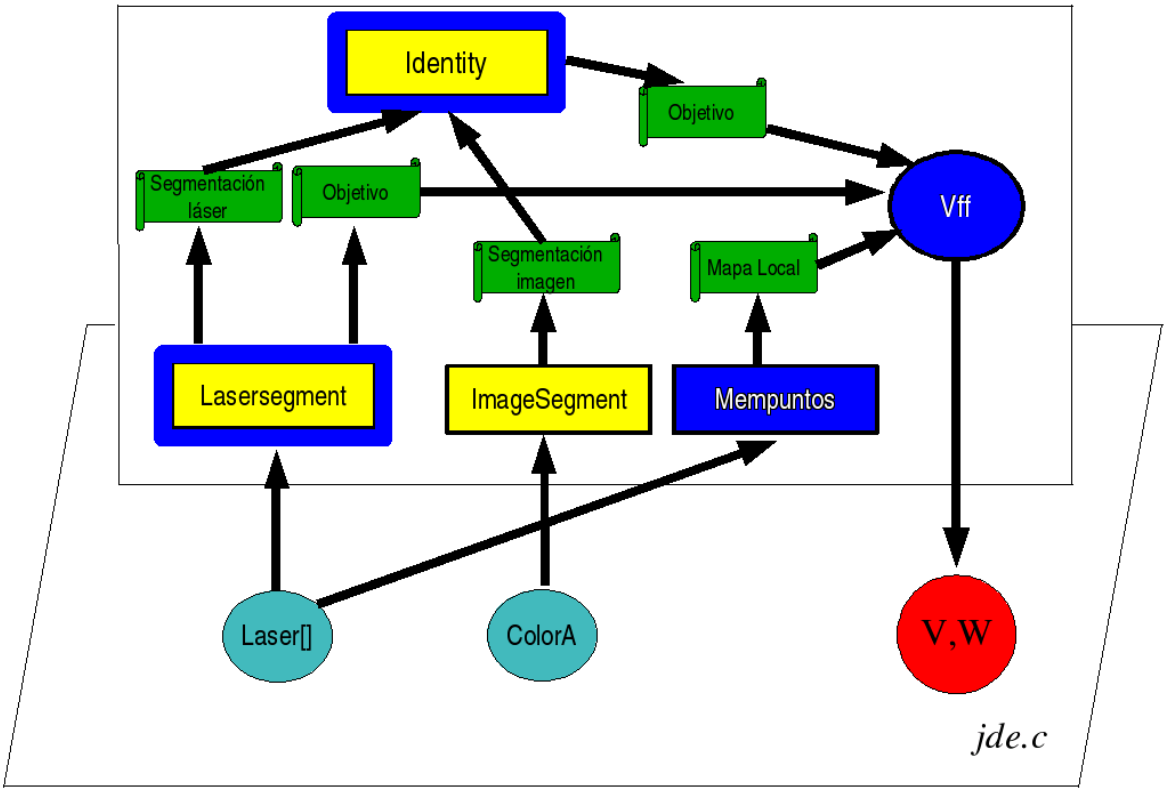



Figura 4.2: Diseño de la aplicación sobre *jde.c*

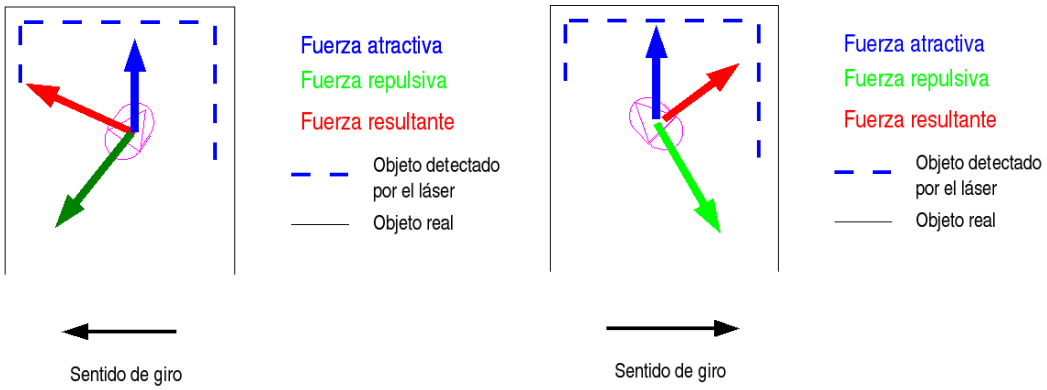
## 4.2. Esquema perceptivo *mempuntos*

Este esquema perceptivo tiene como finalidad la creación de un mapa local para poder navegar sin colisionar con los obstáculos que se le puedan presentar. Además con este esquema conseguimos un mayor alcance y tener presente objetos como los que dejamos atrás. Para ello recoge, resume y almacena la información de ocupación existente en el entorno cercano al robot.

En un principio no se pensó en llevar a cabo este esquema, ya que en el esquema donde está implementado el algoritmo de navegación local obteníamos los datos del entorno directamente del sensor láser. Sin embargo al obtener los datos directamente y no ir almacenando los datos se nos presentó el problema de las oscilaciones.

El algoritmo de navegación local se basa en la técnica del *vff* [Borenstein, 1989], descrita brevemente en el capítulo 1, donde los objetos dan lugar a una fuerza repulsiva que aleja al robot de éstos, la suma de esta fuerza junto con otra fuerza, atractiva, genera una fuerza resultante cuya orientación es la que tiene que conseguir el robot

para evitar los obstáculos y acercarse al robot. Al basarnos sólo en el láser instantáneo y estar cerca de un obstáculo obtendremos una fuerza resultante (figura 4.3(a)). En el momento que el robot empiece a girar dejaremos de obtener parte de esa información de dicho obstáculo, ya que el láser sólo obtiene información de los 180 grados delanteros, al mismo tiempo que vemos una nueva parte del obstáculo por lo que la fuerza repulsiva cambia y cambia la fuerza resultante. Llegará un determinado momento donde la fuerza quede al lado contrario de hacia el lado donde giraba el robot (figura 4.3(b)). Éste cambiará el sentido de giro para llegar a la nueva resultante. Por lo tanto el robot estará oscilando todo el rato para intentar conseguir la orientación marcada por la fuerza resultante.




(a) (b)

Figura 4.3: Problema oscilatorio del algoritmo de navegación local

Una vez comentados la finalidad y motivos por los que se ha desarrollado este esquema veremos un poco más a fondo cómo está implementado. Para poder leer los datos obtenidos del sensor *láser[]* tendremos que leer de la variable láser ofrecida por la plataforma*jde.c*. Así consultando esta variable obtendremos los puntos que iremos utilizando para crear nuestro mapa local. Dicho mapa local se pone a disposición de todos los esquemas en la matriz *memorialocal[]*, que está formada por dichos puntos, tiene un tamaño de 750 posiciones, y está definida por la siguiente estructura:

```
typedef struct {
    float x;
    float y;
    float theta;     /* relatvia al robot*/
    float distancia; /* relatvia al robot*/
    int vida;
}Cmapa;
```

`X` e `y` nos darán la posición actual del punto en el mundo en milímetros, `theta` la orientación del punto con respecto del robot y `distancia` la longitud en milímetros del punto al robot. Este esquema posee un reloj lógico el cual va aumentando una unidad con cada iteración del esquema, así a cada punto se le da un tiempo de vida, que se corresponde con la variable de la estructura anterior: `vida`.

En cada iteración del esquema, realiza iterativamente los siguientes tres pasos: eliminación de los puntos caducados, inserción de nuevos puntos y eliminación de puntos inconsistentes.

Al inicio de cada iteración es preciso actualizar la orientación y la distancia de los puntos con respecto del robot, ya que éste se esta moviendo y girando constantemente. Si no fuera así cuando un obstáculo sea detectado a una determinada distancia, como por ejemplo dos metros, y el robot se vaya acercando, dicho objeto se encontraría para el robot a la misma distancia, dos metros, cuando realmente está más cerca, y el robot podría colisionar con el objeto.

Así se eliminaran los puntos cuyo tiempo de vida sea menor que el reloj lógico del esquema. Además el tiempo de vida de cada punto hemos hecho que dependa de la distancia al robot. Así cuando el robot tenga que evitar un obstáculo, al estar muy próximo a él tarda un poco más en evitarlo por lo que los puntos de ese objeto tienen un tiempo de vida mayor. En concreto, los puntos que estén a una distancia menor de 750 milímetros del robot tendrán más tiempo de vida.

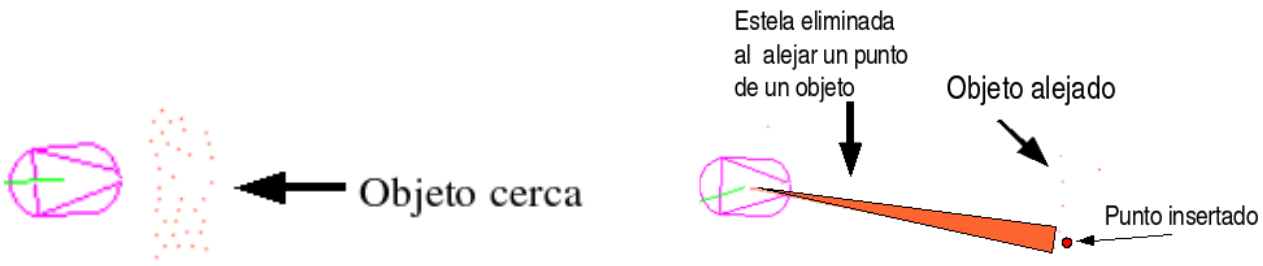
Antes de insertar un punto en la estructura anterior, debemos de comprobar si existen puntos similares, para evitar redundancia. Para ello como tenemos un mapa relativamente pequeño y dado la dificultad de que los puntos coincidan exactamente, un punto se considera igual a otro, si se encuentra dentro de una distancia de similitud. Por ejemplo si tenemos un punto en la posición 1000,1000 y obtenemos un nuevo punto

con la posición 1015,978 ese punto se considerará el mismo, y en vez de insertarse lo único que se hará es actualizar el tiempo de vida del punto. Si el punto que estamos comprobando no tienen ningún punto similar lo insertaremos en un lugar vacío de la memoria de puntos. Pero, si el mapa local está lleno, insertaremos éste en la posición de aquel punto que se encuentre más lejos del robot.

La distancia de similitud entre dos puntos nos introduce el concepto de densidad dentro del mapa local. Así dicha distancia entre puntos es más pequeña para objetos que estén cerca del robot. De este modo tendremos una mayor densidad de puntos en zonas cercanas al robot (figura 4.4) y por tanto objetos cercanos tendrán mayor número de fuerzas repulsivas consiguiendo que la resultante cambie de forma que se evite el obstáculo.


Figura 4.4: densidad de puntos del mapa local

De igual modo que tenemos un mecanismos para insertar puntos en el mapa local, también tenemos que tener un mecanismo para borrarlos. Si no fuera de este modo y nos pusieran un objeto delante del robot e inmediatamente después lo quitaran el robot seguiría viendo un objeto donde no lo hay hasta que se agotara el tiempo de vida de sus puntos. Por lo tanto siempre que insertamos un punto en la memoria comprobaremos que no hay puntos de orientación parecida a una distancia menor, si es así eliminaremos esos puntos (figura 4.5).




(a) Objeto cercano al robot

(b) Al atrasar el objeto se produce la elminación de los puntos del mapa local

Figura 4.5: (a) Eliminación d estela en el mapa local

## 4.3. Esquema perceptivo *lasersegments*

Este esquema se encarga de buscar segmentos rectos dentro de los datos proporcionados por el láser, y de buscar una estructura con forma de pasillo para poner a disposición del esquema actuador el objetivo, en el comportamiento de búsqueda.

De esta manera se consigue una representación más compacta del entorno, permitiendo identificar estructuras como por ejemplo pasillos. Para esto nos apoyamos en un algoritmo abductivo [Gomez, 2002], que tiene como finalidad obtener una representación de los objetos cercanos al robot. Este esquema lo usamos tanto para navegar, en el comportamiento de búsqueda, como para percibir el subestímulo de profundidad en la parte de identificación.

El algoritmo de este esquema ha sido también implementado tomando como entrada la memoria de puntos. Sin embargo por cuestiones de robustez en la identificación, como podremos ver en el capítulo 5, hemos tomado como entrada directa los datos obtenidos por el sensor láser.

Antes de explicar el algoritmo necesitamos ordenar los puntos de entrada angularmente respecto la posición actual del robot. Al utilizar el láser tenemos todas los puntos ordenados. Sin embargo si utilizamos la memoria de puntos debemos ordenar el mapa local ya que tiene todos sus puntos desordenados , en lo que a orientación se refiere. Pero antes de esa ordenación realizaremos una copia interna de la memoria de puntos, ya que los esquemas al ser asíncronos trabajarían sobre la misma matriz

y esto daría lugar a condiciones de carrera. Una vez copiada la memoria de puntos ordenamos la memoria y ahora sí estamos en condiciones de llevar a cabo el algoritmo de segmentación.

Este algoritmo calcula la pendiente entre el primer y último dato del mapa local, para posteriormente comprobar si todos los puntos, con una determinada tolerancia, se encuentran dentro de esta recta hipotética. Para dar por buena esa recta hipotética, y por tanto tomarla como segmento válido, haremos dos comprobaciones. Primero comprobamos que los puntos intermedios no esta más lejos que una determinada franja de tolerancia 4.6(b) (25 centímetros) de la recta hipótesis. Segundo comprobamos que la distancia de un punto con su predecesor es menor a un determinado valor, en nuestro caso 25 centímetros. Si algún punto no cumple las dos reglas anteriores se descarta el segmento, y se vuelve a hipotetizar la recta entre el punto donde se rompió la hipótesis y el último dato del láser o la memoria de puntos. Esto se repite hasta encontrar un segmento válido. Una vez encontrado un segmento se vuelve al principio del algoritmo y se establece la recta hipotética entre el primer punto y el punto anterior al útlimo segmento aceptado como válido, obteniendo así todos los segmentos del entorno.

En la figura 4.6(b) podemos ver como los puntos superan la distancia permitida a la recta hipotetizada. En la figura 4.6(c) observamos como el primer punto que rompió la hipótesis es el comienzo de la nueva recta hipotetizada. Así el algoritmo seguirá calculando rectas hipotéticas hasta encontrar un segmento aceptable (figura 4.6(d)). Después el algoritmo repetirá todos estos pasos y obtendremos como resultado la segmentación del entorno (figura 4.6(e)). Para más detalles de este algoritmo consultar [Gomez, 2002] y el artículo WAF[1].

Una vez conseguida esta representación más compacta del entorno, que utilizaremos en la parte de identificación, buscaremos a partir de ésta una distribución adecuada, en concreto la forma de un pasillo. Asumiremos que el pasillo se define por tener dos largos segmentos paralelos y la distancia entre ellos está entre 1 y 3 metros.

[1] http://www.rvg.ua.es/waf03

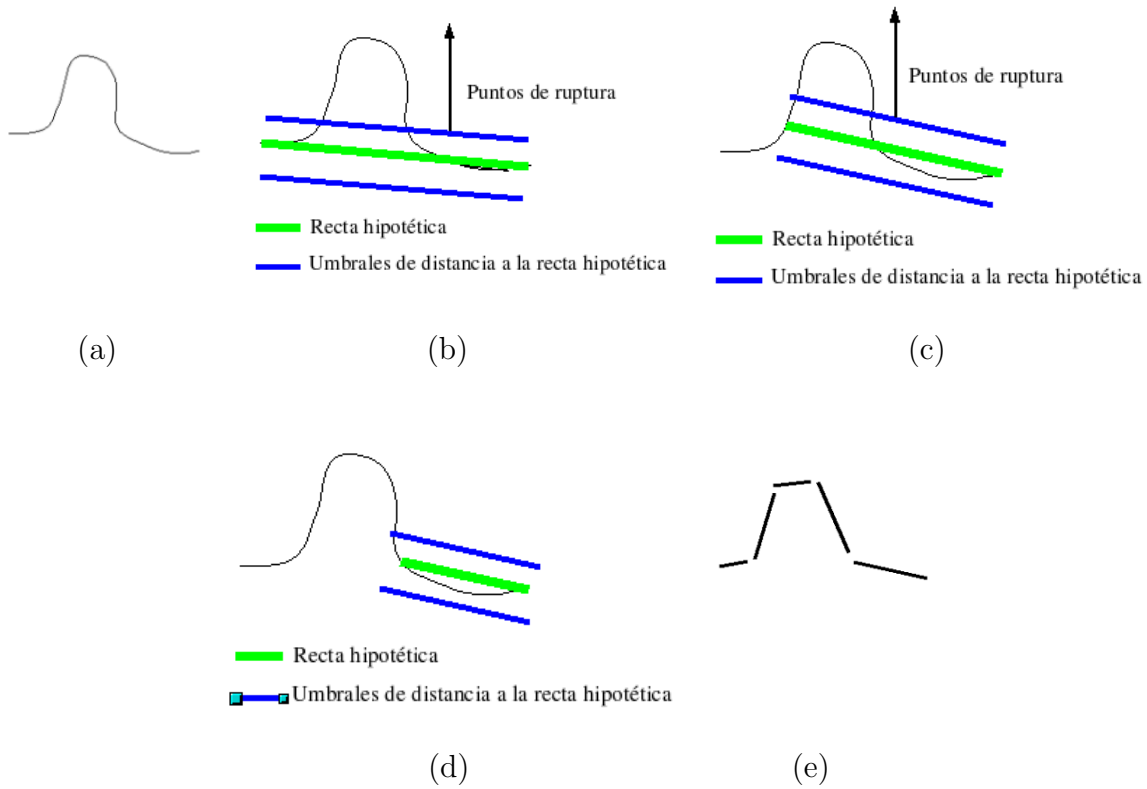
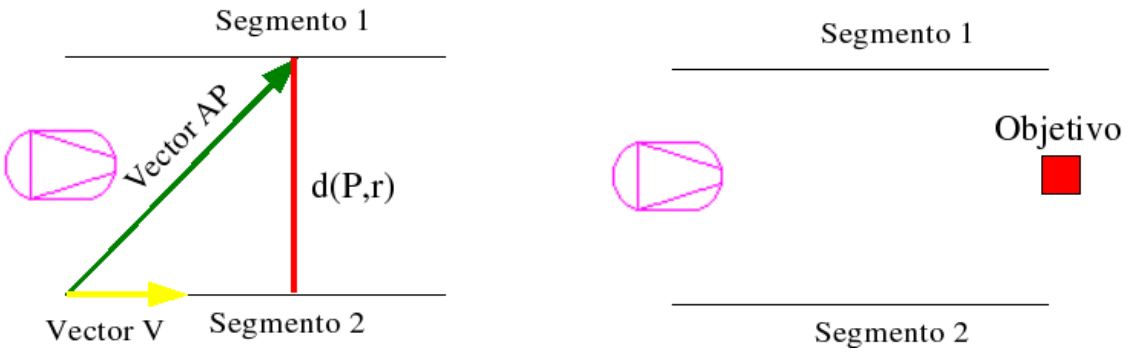



Figura 4.6: Ejemplo de segmentación a partir del mapa loca

Para detectar si los segmentos son paralelos calculamos las pendientes de éstos. Si las pendientes son semejantes dentro de un rango previamente definido, los aceptamos como paralelas. Además, para que sean segmentos que definan un pasillo, exigimos que la distancia entre éstos tiene que estar entre 1 y 3 metros de longitud. Para calcular esta distancia calculamos la proyección ortogonal, siguiendo la ecuación 4.1, del extremo de uno de los segmentos con la recta que contiene al otro segmento (figura 4.7(a).

Una vez que encontradas las líneas paralelas, para determinar el objetivo simplemente hallamos el punto medio, más alejado al robot, de la recta que une los extremos de las líneas paralelas (figura 4.7(b)).

$$\mathrm{d(P,r)} = \frac{|\vec{AP} \quad x \quad \vec{v}|}{|\vec{v}|} \tag{4.1}$$




(a) (b)

Figura 4.7: (a) Cálculo de la proyección ortogonal de un punto sobre una recta (b) Cálculo del objetivo

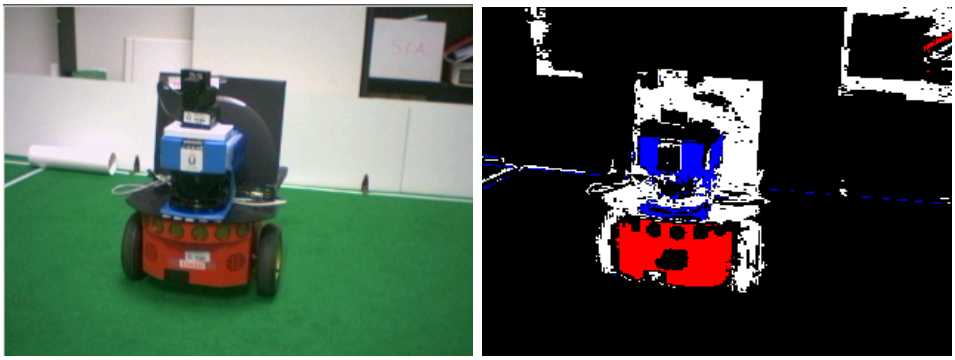
## 4.4. Esquema perceptivo imagesegment

Este esquema perceptivo es el primer paso que se lleva acabo para poder identificar a aquel robot al que se va a perseguir. Su función es procesar las imágenes obtenidas por la cámara, que la plataforma *jde.c* pone a nuestra disposición en la matriz *colorA*, para obtener las agrupaciones de los colores rojo, azul y negro. Sólo se intentarán buscar las agrupaciones de estos tres colores porque son los que predominan en el robot, rojo de la base, azul del sensor láser y negro de las ruedas, portátil y parte delantera del sensor láser (figura 4.8).

El trabajo de este esquema se divide en dos partes que se ejecutan secuencialmente en cada iteración del esquema: la primera se corresponde con el filtrado de color de la imagen, la segunda parte es la agrupación (segmentación) de los colores que hemos filtrado.

### 4.4.1. Filtro de color

En esta etapa pretendemos realzar las partes de la imagen donde se encuentren los colores predominantes en el robot (rojo,azul,negro). Esta parte del esquema es una de las piezas claves del proyecto en el resultado del comportamiento ya que si el filtro no funciona correctamente las acciones que se tomarán sobre el seguimiento del robot serán erróneas. En la figura 4.8 podemos observar cómo a partir de la imagen real filtramos los colores que nos interesarán para la identificación.



(a) (b)

Figura 4.8: (a) imagen real y (b) imagen filtrada por color

La plataforma *jde.c* nos ofrece la imagen en formato RGB. En éste, cada píxel ocupa tres bytes: un byte para la componente roja, otro para la verde y un tercero para la azul. Sin embargo se utiliza para el filtro de color el formato HSI. En dicho formato cada píxel esta formado por tres componentes: matiz (H), saturación (S) e intensidad (I). Su representación geométricamente es un cono como se muestra en la figura 4.9. Como hemos comentado la plataforma *jde.c* nos ofrece la imagen en formato RGB , por lo tanto tendremos que realizar el cambio al formato HSI. El cambio de formato lo realizamos mediante las siguientes ecuaciones:

$$\mathrm{H} = \frac{\frac{-1}{2}(R-G)+(R-G)}{2*\sqrt{((R-G)^2+(R-B)*(G-B))}} \tag{4.2}$$

$$\mathrm{S} = 1 + \frac{3}{R+G+B} * min(R,G,B) \tag{4.3}$$

$$\mathrm{I} = \frac{R+G+B}{3} \tag{4.4}$$

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 como cuando los tres valores son iguales a cero, en estos casos no se aplican estas fórmulas. El siguiente código representa dicho cambio en un pixel:

```
if (((R-G)*(R-G)+(R-B)*(G-B)) < =0)
```

```
   H = -1;
if ((R+G+B) == 0.0)
   S=1.0;
```

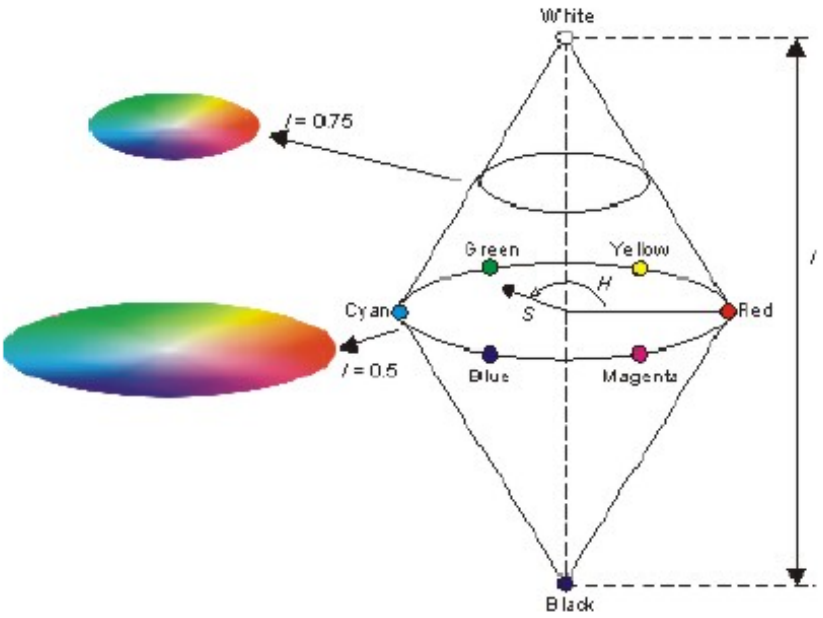
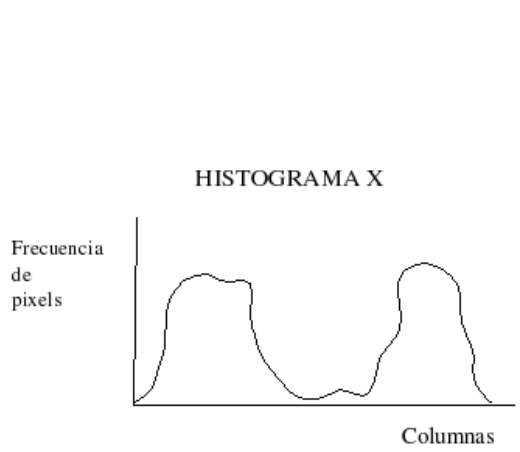


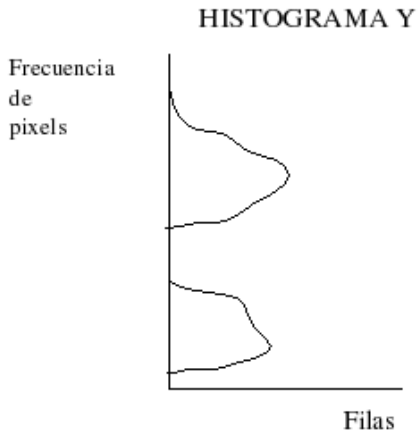

Figura 4.9: Representación del formato HSI

### 4.4.2. Segmentación de la imagen

La segmentación es un proceso que consiste en dividir una imagen digital en regiones homogéneas con respecto a una o más características, en nuestro caso el color, con el fin de facilitar su posterior análisis y el reconocimiento de objetos. La técnica que se hemos empleado para segmentar, está basada en histogramas. Un histograma es un diagrama de barras que contiene frecuencias relativas a algunas características. En nuestro caso por cada color que queramos segmentar, tenemos dos histogramas: uno para las columnas, y otro para las filas. En el histograma para las columnas(figura 4.10(a)), éstas están representadas en el eje de abscisas y en el eje de ordenadas la frecuencia con la que aparece un pixel de un determinado color. El histograma de las filas (figura 4.10(b)) es igual, pero con la diferencia que en el eje de abscisas estarán representadas las filas en vez de las columnas.




(a) Histograma X correspondiente al color rojo de la figura 4.13(b)




(b) Histograma Y correspondiente al color rojo de la figura 4.13(b)

Figura 4.10: A la izquierda el histograma de las filas, y a la derecha el histograma de las columnas

Para poder realizar segmentación nos hemos basado en la técnica del segundo umbral [Martín, 2002] (figura 4.11). La técnica del segundo umbral pasa dos umbrales a cada histograma (filas y columnas), con el primero se eliminan los puntos ruidosos y si se pasa el segundo umbral, partiendo de los puntos que superaron el primero, se crea el segmento. En la figura 4.11 podemos ver cómo tres zonas pasan el primer umbral, sin embargo una de ellas no pasa el segundo umbral por lo que no dará lugar a ningún segmento. Al utilizar este algoritmo, en entornos con colores parecidos, nos dimos cuenta que cuando un objeto coincide con otro objeto en filas y columnas como podemos ver en la figura 5.8, es segmentado todo como si tratase del mismo objeto.
En este proyecto hemos mejorado la técnica del doble umbral, para ello cada vez que obteníamos un segmento comprobamos que el número de pixels de este en relación a su tamaño superaba un porcentaje previamente definido, si no es así procedemos a la segmentación dentro de ese mismo segmento, es decir volvemos a pasar la técnica del segundo umbral a dicho segmento. A esta técnica la hemos denominado: segmentación recursiva, y ajusta el contorno de los objetos mejor que la técnica del doble umbral, como veremos en el capítulo de experimentos.

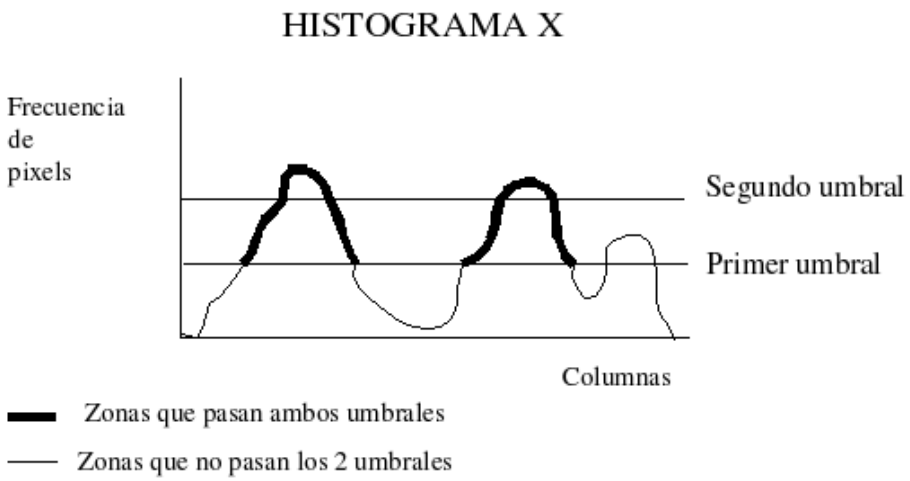



Figura 4.11: Histograma con doble umbral

Para obtener los límites de cada zona, almacenamos por un lado el comienzo y el final de las distintas zonas obtenidas al pasar el segundo umbral en el histograma de las columnas, del mismo modo lo haremos para el histograma de las filas. Usando conjuntamente las zonas que superan ambos umbrales de las filas y las columnas, obtendremos unos segmentos rectangulares de color en la imagen (figura 4.12). Estos segmentos no son definitivos, ya que como vemos en la figura 4.12 se pueden llegar a crear falsos segmentos por lo que hay que comprobar que superan un número mínimo de píxeles.

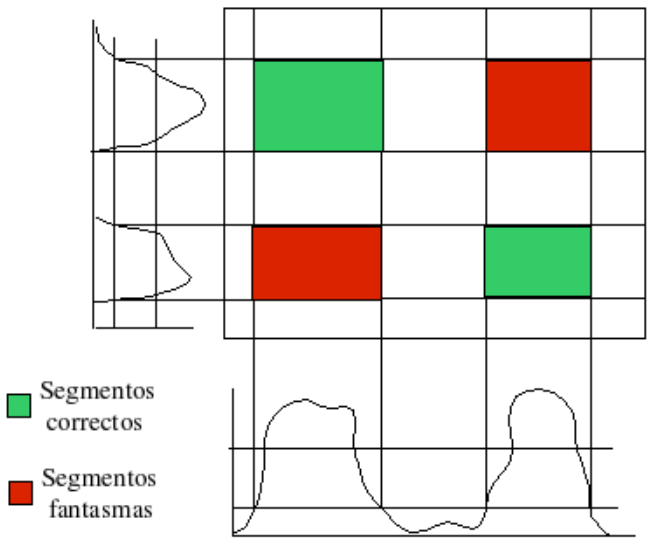
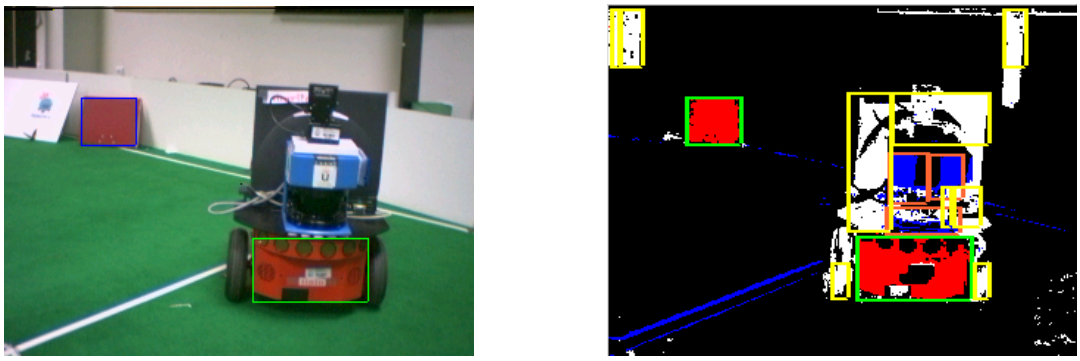



Figura 4.12: enventando de colores

Una vez realizado todo este proceso, obtendremos como resultado los segmentos de los colores rojo,azul y negro como podemos ver en la figura 4.13. Este esquema ofrece al resto de esquemas los segmentos de la imagen en la matriz *segidentity[]*, que tiene la siguiente estructura:

```
typedef struct {
     int c_fila;      /* comienzo del segmento en las filas */
     int c_columna;   /* comienzo del segmento en las columnas */
     int f_fila;      /* fin del segmento en las filas */
     int f_columna;   /* fin del segmento en las columnas */
     int color;       /* color del segmento */
     int puntuación;
};
```



(a) (b)

Figura 4.13: (a) Imagen inicial (b) Imagen segmentada

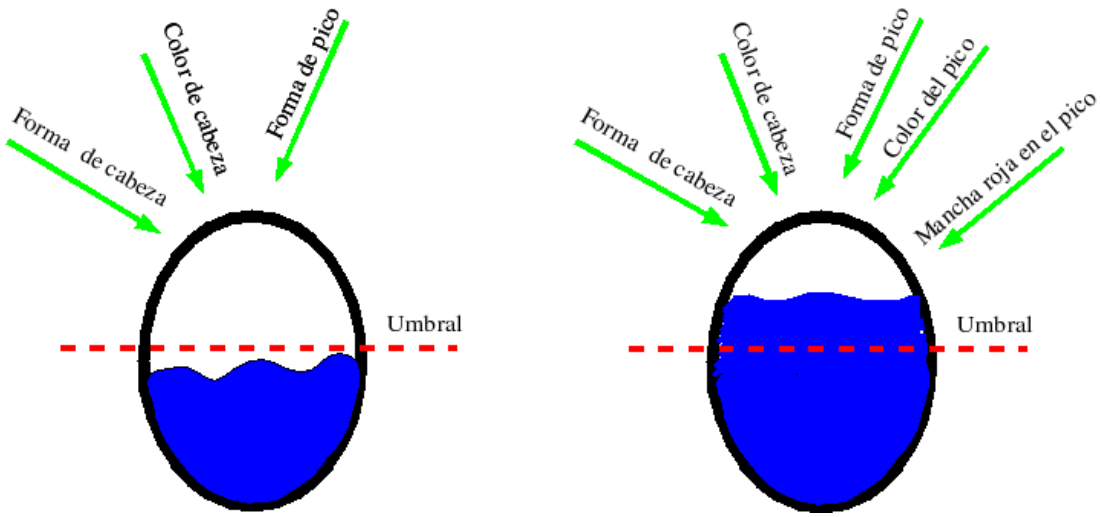
## 4.5. Esquema identity

Este esquema recibe como entrada los segmentos de color obtenidos por el esquema *imagesegment* y los segmentos del entorno obtenidos por el esquema *lasersegment*. A partir de éstos extraemos las características de cada segmento y llegaremos a la identificación o no del robot congénere.

Para realizar esta identificación, en vez de basarnos en el reconocimiento de patrones clásico, hemos diseñado una nueva técnica para conseguir una identificación más robusta. Esta nueva técnica se basa en la *etología*, más concretamente en el *principio de suma heterogénea de subestímulos* [Lorenz, 1978]. Este principio dice que si la suma

de un conjunto de subestímulos supera cierto umbral se producirá el desencadenamiento de un comportamiento. Con este principio conseguimos una identificación a partir de ciertos subestímulos sencillos, sin tener que hacer una reconstrucción completa del mundo.

Por ejemplo, en el comportamiento de apertura de pico de las crías de gaviota cuando viene el progenitor para alimentarlas intervienen muchos factores, la forma y color del pico que observa la cría, el color y forma de la cabeza, una mancha roja en el pico etc. Así si el efecto acumulativo de todos los subestímulos supera cierto umbral la cría abrirá el pico. A modo de ejemplo podemos ver la figura 4.14 donde al principio tres subestímulos no producen el comportamiento, luego la percepción de dos subestimulos más produce el desencadenamiento del comportamiento al superar el umbral.




(a) La suma de subestímulos no desencadena el comportamiento

(b) La suma de subestímulos desencadena el comportamiento

Figura 4.14: Principio de suma heterogénea de estímulos

Utilizando este principio y usando como entrada los segmentos de color y del láser, llegaremos a la identificación del congénere. En determinados comportamientos existen condiciones imprescindibles, es decir, necesitan de ese subestímulo obligatoriamente para que se desencadene el comportamiento. Nosotros tendremos una condición imprescindible: los segmentos de color rojo, ya que es un invariante del robot congénere independientemente de la posición desde que sea observado. Así extrayendo los su-

bestímulos que se acumulan sobre los segmentos rojos, llegaremos a la identificación o no del robot. La combinación de subestímulos simples permite conseguir una gran robustez y discriminación. Los subestímulos son los siguientes:

1. El segmento rojo sea más ancho que alto. Este subestímulo se corresponde con con la base del robot ya que es más ancha que alta desde todos los ángulos que se pueda mirar al robot. Esta característica incrementa la suma de los subestímulos en dos puntos.

2. Encima del segmento rojo hay un segmento azul. Este subestímulo se corresponderá con que el sensor láser que está encima de la base del robot. Para contabilizar esta característica, exigiremos que el segmento azul sea más estrecho que el rojo, además estará entre la columna comienzo y final del segmento rojo , y el fin de la fila del segmento azul estará en un rango de diez píxeles por arriba del segmento rojo.Esta característica produce un incremento en la suma de subestímulos de dos puntos.

3. Encima del segmento rojo hay un segmento negro. Este estímulo se corresponderá con que el portátil o la parte negra del sensor láser este encima de la base del robot. Para contabilizar esta característica, exigiremos que el segmento negro está entre la columna comienzo y fin del segmento rojo, y el fin de la fila del segmento negro estará en un rango de diez píxeles por arriba del segmento rojo.Esta característica produce un incremento en la suma de subestímulos de dos puntos.

4. El segmento rojo de sensación de profundidad. Por cada segmento rojo, hallaremos la posición del pixel central del segmento en el mundo, utilizando para ello la biblioteca *PROGEO*, y la segmentación del entorno que es ofrecida por el esquema *lasersegment*.
   Entre el pixel central del segmento rojo y la cámara generamos un rayo visual. Así cuando el rayo visual corte a un segmento producido por el láser, obtendremos la posición en el mundo, donde se encuentra el segmento rojo. En la figura 4.15 podemos ver un ejemplo a vista de pájaro de cómo cortan estas rectas a distintos segmentos, que se corresponde con la imagen de la figura 4.13(b) y donde se puede ver que a la derecha del robot se encuentra el congénere .
   Una vez obtenida la posición donde se encuentra el segmento rojo, estudiaremos si da "sensación de profundidad". El segmento, que es cortado por el rayo visual, tiene que ser menor a 600 milímetros (ancho del robot), además este segmento no tiene que tener cerca ningún otro segmento para que así produzca "sensa-

ción de profundidad ". Esta característica produce un incremento en la suma de subestímulos de ocho puntos.

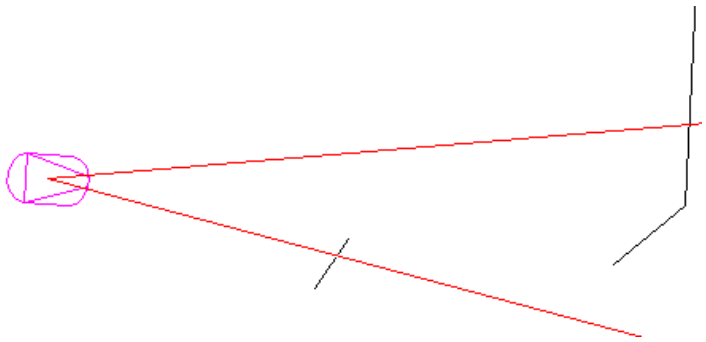


Figura 4.15: correlación visión láser

Para que un segmento rojo sea del robot, y por tanto sea perseguible, los indicios que se acumulan deben ser mayor o igual a 12 puntos, además todos los segmentos rojos que superen los 6 puntos, serán considerados como un posible candidato si se encuentran a más de dos metros y medio de distancia. Si no tenemos el robot identificado, pero sí candidatos nos acercaremos a estos, no más de dos metros y medio, para ver si son el robot o no.

Este esquema ofrece a otros esquemas la posición en el mundo del congénere, si es identificado, y en caso de no hacerlo la de el candidato con más puntuación, en la siguiente estructura:

```
typedef struct {
     Tvoxel punto3D
     int identificado;
     int candidato;
}
```

Donde la variable `identificado` nos dice si el robot es identificado o no, si el robot no es identificado la variable `candidato` nos dice si el robot es un candidato o no. La variable `punto3D` nos dará la posición en el mundo donde se encuentra el robot o el candidato.

## 4.6. Esquema actuador *VFF*

Este esquema se encarga de comandar el robot, por lo tanto es un esquema de actuación. Aquí desarrollamos la navegación local en nuestro proyecto que implementa la técnica de Campo de Fuerzas Virtuales o *Virtual Force Fields (VFF)*, [Borenstein, 1989].

Para poder llevar acabo la técnica del VFF, necesitamos saber dónde se encuentran los objetos en el mundo, que lo conseguiremos con el mapa local obtenido por el esquema *mempuntos*, y con el objetivo. El objetivo es la posición del robot congénere o algún candidato, o en el caso de que no haya ningún candidato ni el robot, una nueva zona exploración. Además de esto tenemos que producir un movimiento suave.

En este esquema reside la diferencia entre búsqueda y seguimiento. Así si no hemos identificado al robot, estaremos generando el comportamiento de búsqueda. En este caso el objetivo bien es un candidato a ser el robot, o si no hay candidatos el objetivo es dado por el esquema *lasersegment*. Sin embargo, generamos el comportamiento de seguimiento, cuando el robot sea identificado.

Una vez que tenemos el objetivo, ya sea obtenido por el esquema *identity* o por el esquema *lasersegment*, estamos en condiciones de explicar la técnica del *vff* y como se ha conseguido obtener un movimiento suave para poder navegar usando un controlador borroso.

### 4.6.1. 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 hacia 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 de evitar a los obstáculos, y simultáneamente llegar al objetivo de la navegación.

El algoritmo que implementa esta técnia es reactivo ya que en cada iteración calcula todas las fuerzas. Como hemos dicho anteriormente, tomamos como entrada los obstáculos del entorno y el objetivo. Los obstáculos serán los puntos del mapa local, que se encuentran en la matriz *memorialocal[]*, y que serán los que ejerzan las fuerzas

repulsivas sobre el robot. Las fuerzas virtuales repulsivas se calculan como el cociente de una constante de repulsión entre la distancia del robot al objeto. Esta 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. El objetivo producirá la fuerza atractiva. Ésta se calcula cómo el cociente de una constante atractiva entre la distancia del robot al objeto, que será mayor cuanto más cerca estemos del objetivo. La fuerza resultante será la suma de ambas fuerzas y proporcionará al esquema la dirección a seguir.

$$\text{Xrepulsiva} = \sum_{i=0}^{n} (Krepulsiva/distancia^2) * \cos(\theta) \tag{4.5}$$

$$\text{Yrepulsiva} = \sum_{i=0}^{n} (Krepulsiva/distancia^2) * \sin(\theta) \tag{4.6}$$

$$\text{Xatractiva} = (Katractiva/distancia) * \cos(\theta) \tag{4.7}$$

$$\text{Xatractiva} = (Katractiva/distancia) * \sin(\theta) \tag{4.8}$$

$$\text{X}_\text{r}\text{esultante} = Xatractiva + Xrepulsiva \tag{4.9}$$

$$\text{Yresultante} = Yatractiva + Yrepulsiva \tag{4.10}$$

Donde $n$ es el tamaño correspondiente con el mapa local, *Katractiva* la constante de atracción, *Krepulsiva* la constante de repulsión, *theta* la orientación del punto, atractivo o repulsivo, respecto del robot. La constante Katractiva en este proyecto puede tomar dos valores, un valor mínimo o un valor máximo. Asi cuando estemos cerca de un objeto utilizaremos el valor mínimo para dar más importancia a las fuerzas repulsivas y así evitar el obstáculo.

Podemos ver un ejemplo de esto en la figura 4.16, donde la línea azul es la fuerza atractiva, generada por el destino, la línea verde es la fuerza repulsiva, producida por los puntos del mapa local y la línea roja es la fuerza resultante obtenida al sumar las fuerzas anteriores.

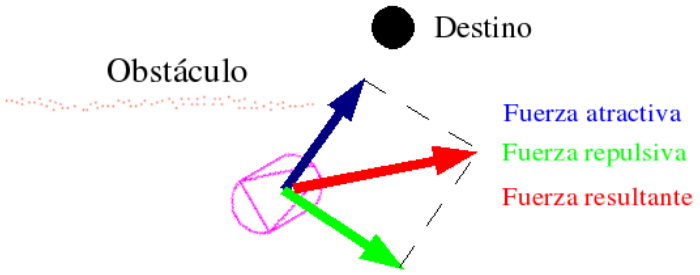



Figura 4.16: Visualización de la fuerzas del vff

Esta forma de navegación local nos garantiza la evitación de obstáculos tanto estáticos como dinámicos. Como puedan ser personas o paredes, porque ambos son detectados por el sensor láser e insertados en el mapa local.

### 4.6.2. Controlador borroso

Una vez obtenida la orientación de la fuerza resultante, usaremos para comandar las velocidades del robot, un controlador borroso [Isado, 2005]. A grandes rasgos el controlador borroso consigue suavidad en el movimiento, y que éste sea rápido. Además 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*, como se comentó en el capitulo 3. Dicho controlador acepta como entradas:

1. texttt{á}ngulo: Diferencia entre el ángulo actual del robot y el de la resultante. Para hallar el angulo de la fuerza resultante, puesto que ésta se expresa en el sistema de referencias del robot hemos hecho el siguiente cálculo trigonométrico:

$$\theta = \arctan(Yresultante, Xresultante) \tag{4.11}$$

2. v-Actual: La velocidad lineal actual

3. w-Actual: La velocidad angular actual

4. peligro: Advierte de la cercanía de un obstáculo. Esta variable de entrada puede tomar diferentes valores. Valdrá 3 cuando el objeto esté a menos de 800 mm del robot frontalmente, o a 300 mm lateralmente. Valdrá 2 cuando el objeto esté a una distancia de un metro frontalmente. Tomará el valor de 1 cuando esté a una distancia de 400 mm lateralmente y cero cuando no haya cerca objetos. Con esto

conseguiremos reducir la velocidad cuando se vaya acercando un objeto y cuando la variable valga 3 el robot parará.

El controlador borroso define unas etiquetas sobre las variables de entrada y de salida. Estas etiquetas para la diferencia angular y la velocidad angular de entrada y salida, son las siguientes:

```
etiqueta velocidad_angular alta_pos = 20 35 45 50
etiqueta velocidad_angular media_pos = 5 15 25 35
etiqueta velocidad_angular baja_pos = 0 5 5 10
etiqueta velocidad_angular nula = 0 0 0 0
etiqueta velocidad_angular baja_neg = -10 -5 -5 0
etiqueta velocidad_angular media_neg = -45 -30 -15 -5
etiqueta velocidad_angular alta_neg = -50 -45 -35 -20
```

En la figura 4.17 podemos ver gráficamente las etiquetas para la velocidad angular. Donde las entradas están normalizadas entre cero y uno. Los valores de salida se corresponden con las etiquetas anteriores. En el capítulo de experimentos veremos más detalladamente como funciona el controlador borroso.
Las reglas que determinaran las variables de salida para la velocidad angular son:

```
IF ( angulo = nulo ) THEN ( velocidad_angular = nula )
IF ( angulo = bajo_pos ) THEN ( velocidad_angular = baja_pos )
IF ( angulo = medio_pos ) THEN ( velocidad_angular = media_pos )
IF ( angulo = alto_pos ) THEN ( velocidad_angular = alta_pos )
IF ( angulo = bajo_neg ) THEN ( velocidad_angular = baja_neg )
IF ( angulo = medio_neg ) THEN ( velocidad_angular = media_neg )
IF ( angulo = alto_neg ) THEN ( velocidad_angular = alta_neg )
```

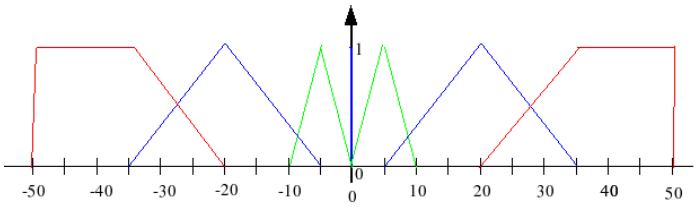



Figura 4.17: Gráfica de la velocidad angular en el controlador borroso

## 4.7. Interfaz gráfica

La interfaz gráfica esta implementada en el esquema *guipursuit*. Este esquema, es un esquema de servicio, que nos sirve para poder depurar nuestro proyecto. Esta interfaz está programado con la biblioteca *Xforms*. Este esquema se encarga de visualizar los resultados perceptivos de los distintos esquemas de nuestra aplicación. Al ser un esquema más se ejecuta iterativamente, y en cada iteración comprueba el estado de los elementos de la interfaz y muestra en pantalla los resultados pedidos.

Pulsando el botón adecuado podremos ver sobre la aplicación la imagen, tanto real como filtrada, las fuerzas generadas por el *VFF*, las rectas entre la cámara y el plano de proyección, el mapa local , los segmentos obtenidos a partir de este mapa, las medidas de distintos sensores y comandar los actuadores. Además podemos poner en marcha o dormir el comportamiento programado. Un ejemplo de esta interfaz lo tenemos en la figura 4.18.

Esta interfaz se ha realizado a partir de la interfaz *guixforms*, usada como plantilla. Sobre dicha plantilla hemos modificado y añadido ciertos elementos para adaptarla a nuestras necesidades. Los botones genéricos que hemos utilizado y ya existían en *guixforms* son:

- Láser (muestra u oculta las medidas sensoriales recogidas por el láser),Robot(muestra u oculta el robot), ColorA (muestra u oculta la imagen obtenida por la cámara),
- Motors (permite que mandamos ordenes a los actuadores, si no estuviera activada el robot nunca se movería).

Los botones que hemos añadido para adaptarlo a nuestras necesidades son:

- Mempuntos: visualiza los puntos del mapa local
- Vff: Este boton activa el esquema dónde está implementado el algoritmo de navegación local. Así al pulsar este boton podremos ver las fuerzas atractiva, repulsiva y resultante en todo momento.
- Objetivo: Este botón ha sido muy útil para poder depurar, ya que nos permite generar el objetivo de varias maneras.

    1. Con el botón ratón podíamos generarlo al hacer un click de ratón sobre el canvas.

2. Con el botón repulsivas, el algoritmo obtendría como fuerza resultante las fuerzas repulsivas.
3. Con el botón deambular nos permite realizar la ejecución típica de esta aplicación.

- segmentar laser: Activa el esquema *lasersegment* y nos muestra en el canvas los segmentos calculados
- Filtro: Activa el esquema *imagesegment* y nos mostrará la imagen filtrada por color y segmentada.
- identificación: Activa el esquema *identity*, y nos mostrará el rayo visual entre la cámara y todos los segmentos rojos. Además muestra en la imagen real un cuadrado blanco alrededor del robot identificado, o un cuadrado azul alrededor de los candidatos. Así podemos comprobar si el algoritmo de clasificación distingue entre el robot congénere y otras cosas que aparecen en el entorno del robot.

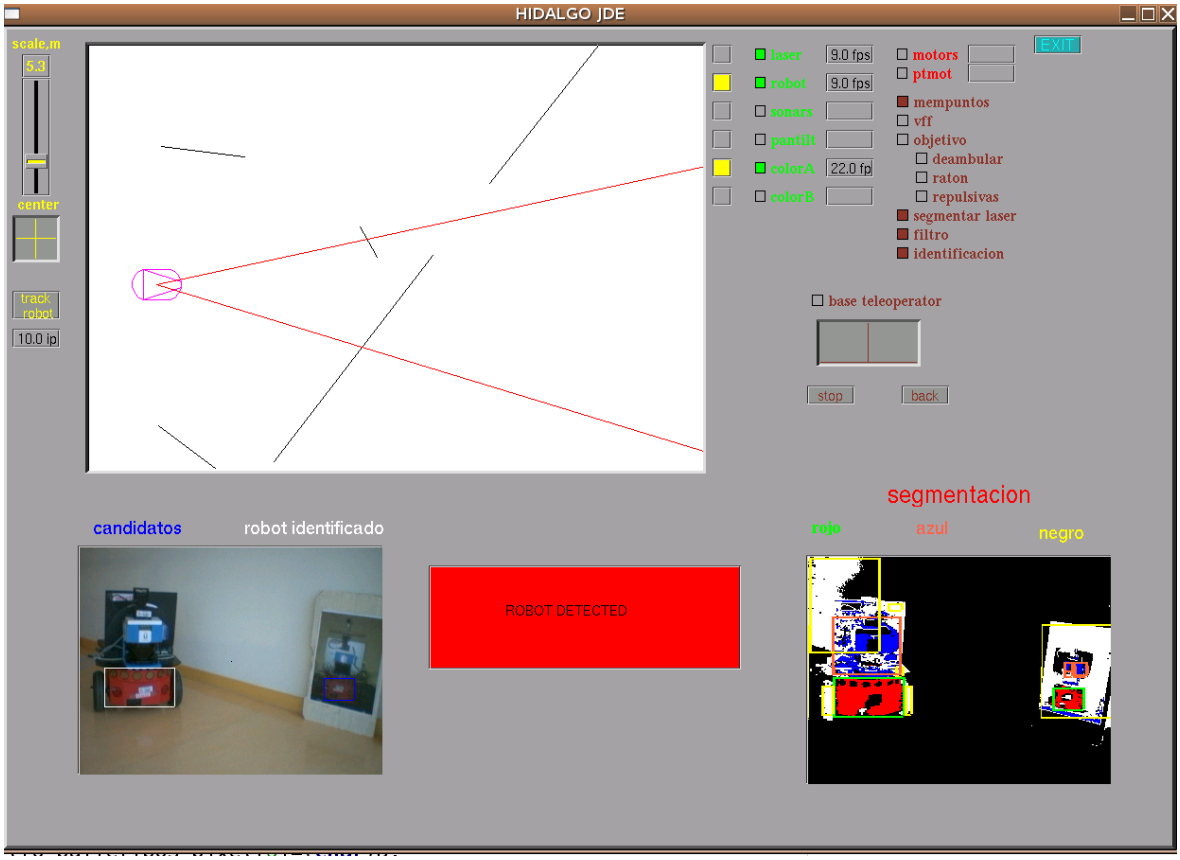



Figura 4.18: Interfaz gráfica de la aplicación

# Capítulo 5
# Resultados experimentales

En el capítulo anterior hemos comentado la solucción final que hemos programado. En este capítulo explicaremos qué pruebas y experimentos hemos realizado para validar y mejorar la implementación en el camino hacia esta solución, según distintos prototipos desarrollados iterativamente, como vimos en el capítulo 2. Comenzaremos detallando la ejecución típica del comportamiento que hemos generado. Seguidamente explicando los experimentos realizados en la navegación. Por último comentaremos las pruebas realizadas en la parte visual del proyecto.

## 5.1. Ejecución típica del comportamiento

Una vez explicados todos los componentes de la aplicación en el capítulo anterior, pasamos a probar el sistema completamente integrado y sobre el robot real.

En la ejecución de todo el sistema integrado, estarán todos los esquemas activos. Con esto conseguimos una *secuencia flexible* en el comportamiento, ya que el robot primero busca al robot y cuando lo identifique le seguirá. Sí mientras le sigue se diera el caso de que pierde de vista al congénere, entonces comenzará una nueva búsqueda.

Inicialmente el robot busca a su congénere por los pasillos del departamental, para ello como el esquema *identity* no da ningún resultado, es el esquema *lasersegment* es el encargado de generar un destino al esquema *VFF*, buscando como explicamos en el capítulo anterior una figura semejante a la de un pasillo. Pudimos comprobar que a pesar de haber despachos con las puertas abiertas no suponía ningún problema y navegaba correctamente. Como ejemplo podemos ver las figura 5.5, donde el destino es dado por el esquema *lasersegment* y al llegar a la esquina el robot gira hasta encontrar un nuevo objetivo, y la figura 5.6, donde se evitan obstáculos para buscar al robot, en este caso los objetos son dados al esquema *VFF* mediante el mapa local y el destino es

generado por el esquema *lasersegment*. También probamos a poner objetos de color y tamaño parecido en el pasillo. En la figura vemos como hay un poster del robot *Pioneer* en la pared, al que el esquema *identity* acaba descartando como robot y prosigue la búsqueda, tomando como destino el dado por el esquema *lasersegment*.

(a) (b) (c)

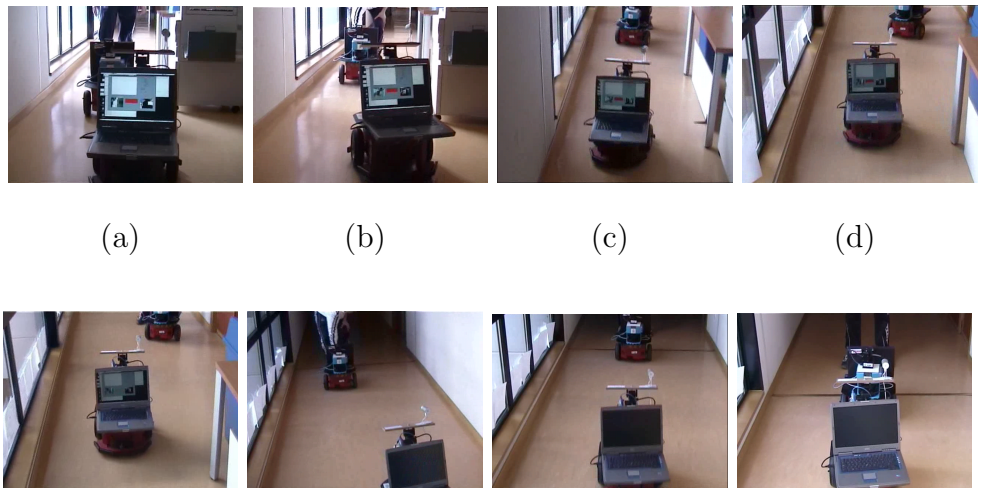
Figura 5.1: Poster del robot Pioneer descartado como robot real

Cuando el esquema *identity* detecta al congénere, se pasa del estado de búscqueda al estado de seguimiento. En este caso el esquema *identity* genera la posición de donde se encuentra el congénere, y da como objetivo al esquema *VFF* dicha posición en el mundo. Si el robot llega a estar muy cerca de su congénere (aproximadamente un metro), por ejemplo porque éste se pueda encontrarse parado, detendremos la marcha hasta que dicho congénere esté a una distancia prudencial para que no se choque. Al realizar ésta prueba cuando el robot se acercaba al congénere demasiado, se acababa perdiendo su imagen en la cámara y por tanto el esquema *identity* no daba ningún destino y el robot pasaba al estado de búsqueda, es decir el destino esta vez será generado por el esquema *lasersegments*. Para evitar ésto, una vez que el congénere estuviera identificado a menos de una distancia determinada ( dos metros ), nos apoyaríamos en el esquema *lasersegment*. Así si lo perdemos de vista, el esquema *identity* comprueba que en la posición donde estaba el congénere hay un segmento por lo que el robot sigue estando en la misma posición, si no fuera así pasaremos al estado de búsqueda. En la secuencia de la figura 5.2 podemos ver como el robot pasa de estar en el estado de búsqueda, al estado de seguimiento ya que se identifica al robot, y al acercarse a éste se detiene a una distancia prudencial para no chocar con el congénere, que en este caso es el objetivo.

(a) Búsqueda (b) Identificación y acercamiento (c) Detección a una distancia prudencial

Figura 5.2: Cambio de estado: de búsqueda a seguimiento

Una vez que el robot está parado delante de su congénere, probamos el comportamiento de seguimiento, generado por el esquema *VFF* y tomando como objetivo la posición en la cual se identifica a ese congénere. Como podemos ver en la secuencia de la figura 5.3, movemos el congénere y el robot persigue a su congénere.



(a) (b) (c) (d)

(e) (f) (g) (h)

Figura 5.3: Seguimiento del congénere

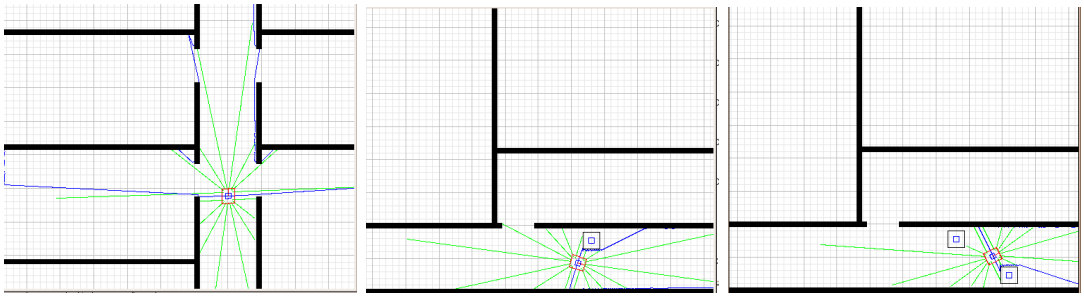
## 5.2. Experimentos en navegación

En esta sección comentaremos las pruebas y experimentos realizadas en la navegación. Nuevamente recordamos que la técnica que utilizamos para la navegación local es la de *VFF*. En la que tendremos cómo destino el robot identificado, generando el com-

portamiento de seguimiento, o un punto aleatorio en el centro del pasillo, generando el comportamiento de búsqueda.

Al comienzo se implementó el sistema de navegación del *VFF* mediante el sensor láser y la corona de sónares traseros, pero los sónares traseros sólo nos proporcionaban 8 medidas por iteración, por lo que dimos un mayor peso a las medidas de los sónares para generar las fuerzas repulsivas que a las medidas del láser. A pesar de esto el robot oscilaba bastante, incluso cuando no tenía obstáculos cercanos y cuando teóricamente debía navegar en línea recta el robot bailaba. Por lo tanto optamos por basarnos sólo en las medidas del láser consiguiendo que el robot no oscilase cuando no tenía objetos cercanos, pero aún así nos encontramos con el problema de las oscilaciones al rebasar objetos, como explicamos en el capítulo 4. Por lo que implementamos la memoria de puntos, y está vez sí conseguimos rebasar los obstáculos. Además como el robot siempre va a moverse hacia adelante, los puntos del mapa local que se encuentren detrás del robot, siempre y cuando estén más de un metro de distancia para no tener el problema de las oscilaciones con objetos cercanos, no son tenidos en cuenta.

Antes de probar el algoritmo de navegación local nos apoyamos en el simulador *Player/Stage*. Una vez comprobado que funciona el algoritmo de navegación pasamos a realizar nuevos experimentos ahora en el robot real. En el simulador, utilizamos un mapa del departamental donde probamos varias situaciones en las que se podría encontrar el robot. Como ejemplo podemos ver la figura 5.6 5.4(a) donde el robot deambula por el pasillo, y donde comprobamos el buen funcionamiento de la selección de un punto aleatorio en el centro del pasillo como destino del esquema *VFF*. En las figuras 5.5(b) y 5.5(c) vemos cómo el robot está deambulando y en su trayectoria tiene que evitar dos obstáculos, percibidos con el sensor láser e insertados en la memoria de puntos, comprobando así que el algoritmo es bastante ágil ante objetos imprevistos.



(a) deambulación por el pasillo

(b) Evitación del primer obstáculo dinámico

(c) Evitación del segundo obstáculo dinámico

Figura 5.4: Pruebas con el algoritmo de navegación el Player/Stage

Realizadas exitosamente las pruebas en el simulador, pasamos a probar el algoritmo en el robot real. Tras varias pruebas vimos que cuando el robot iba a evitar un obstáculo lo conseguía pero le costaba bastante. Esto es debido a que la fuerza resultante varia constantemente en un pequeño ángulo, constando al robot mucho alinearse con ésta y provocando un mínimo movimiento oscilatorio. Dicho problema es provocado porque los puntos del mapa local no están siempre a la misma distancia respecto del robot, ya que este se mueve para evitar el obstáculo. Asi decidimos implementar un controlador borroso, explicado en el capítulo anterior. En las primeras pruebas sobre el robot real hasta que el robot no se alineaba con la resultante el robot tenía una velocidad lineal nula, además la influencia de la resultante sobre la velocidad angular producía cambios bruscos en el sentido de giro. Sin embargo el controlador borroso suaviza la influencia de la resultante en la velocidad lineal y angular, de tal manera que no se note apenas un movimiento oscilatorio. El algoritmo con el controlador borroso nos permitió solucionar dicho problema.

Una vez obtenido un movimiento suave en el contro del robot realizamos pruebas para verificar la correcta navegación. En la figura 5.5 vemos que el robot deambula por el pasillo sin ningún tipo de obstáculo, y dobla perfectamente la esquina. Aumentando la dificultad, como último experimento de la navegación colocamos un obstáculo y como podemos ver en la figura 5.6 el robot la evita . Según rebasa esa caja le colocamos de repente y muy cerca al robot otra caja, la que acaba rebasando perfectamente. Nuevamente según pasa la última caja, le colocamos otra a la que acaba evitando.

(a) Deambulación por el pasillo

(b) Llega a la esquina

(c) Gira para proseguir la búsqueda

Figura 5.5: Búsqueda en las esquinas

(a) (b) (c) (d)

(e) (f) (g) (h)

Figura 5.6: Evitación de obstáculos

Por último, vamos a hablar sobre la velocidad lineal a la que avanza el robot. Al principio empezamos con velocidades relativamente bajas para que en las primeras pruebas si colisionaba con algún obstáculo no lo hiciera violentamente. Una vez conseguida la navegación correctamente fuimos modificando las reglas del controlador borroso para aumentar la velocidad paulatinamente, hasta llegar a alcanzar una velocidad lineal punta de 650 mm/s. Esta velocidad ha sido una mejora objetiva de este proyecto respecto otros l grupo de robótica, entre todos los proyectos de navegación sobre el robot real, la máxima velocidad conseguida era de 400 mm/s. A pesar de tener una velocidad tan alta, en ningún momento hemos comprometido la seguridad, es decir el robot no se choca con ningún obstáculo. Para ello hemos aumentado la distancia de

seguridad, en la que el robot empieza a disminuir considerablemente su velocidad, a un metro mientras que en otros proyectos estaba entorno al medio metro.

## 5.3. Experimentos en identificación

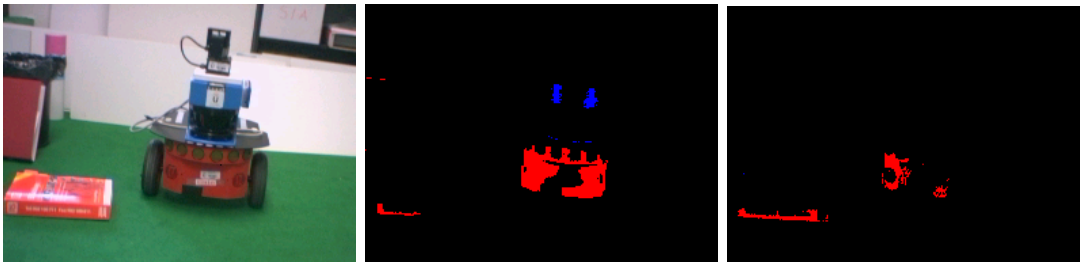
El objetivo a la hora de identificar al congénere, no era sólo su detección, si no conseguir que fuera lo más robusta posible. Así se han hecho pruebas con filtros RGB y HSI, se ha mejorado la técnica de doble umbral en la segmentación, y hemos conseguido una combinación de subestímulos sencillos que no sea posible engañar al robot en la identificación de su congénere, con objetos de color y tamaño parecido, al menos en ninguno de los objetos existentes del departamental 2.

Esta sección la dividimos en tres partes. Primero veremos las pruebas realizadas para procesar correctamente la imagen. Después veremos las pruebas realizadas para sacar el subestímulo de profundidad. Por último comentaremos las pruebas realizadas para el *principio de suma heterogénea de estímulos*.

### 5.3.1. Procesamiento de la imagen

Una parte fundamental a la hora de identificar al congénere, es procesar correctamente la imagen. Si no se consiguiera un buen procesamiento, los resultados a la hora de identificar serían probablemente erróneos. En esta sección comentamos las pruebas realizadas en los filtros de color y en la segmentación para obtener una imagen procesada correctamente.

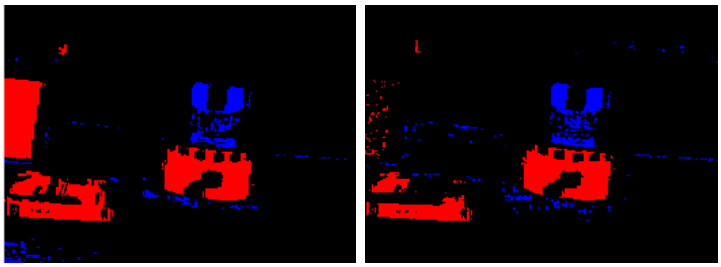
La primera implementación para el filtro de color fue en el formato RGB. Haciendo varias pruebas nos dimos cuenta que éste filtro era muy sensible a los cambios de luminosidad como podemos ver en las figuras 5.7(b) y 5.7(c). Por este motivo implementamos el filtro de color en el formato HSI, que es más robusto a los cambios de luminosidad. Como podemos ver en las figuras 5.7(d) y 5.7(e) los resultados son bastante mejores.



(a) Imagen original

(b) Filtro RGB con luz artificial

(c) Filtro RGB sin luz artificial



(d) Filtro HSI con luz artificial

(e) Filtro HSI sin luz artificial

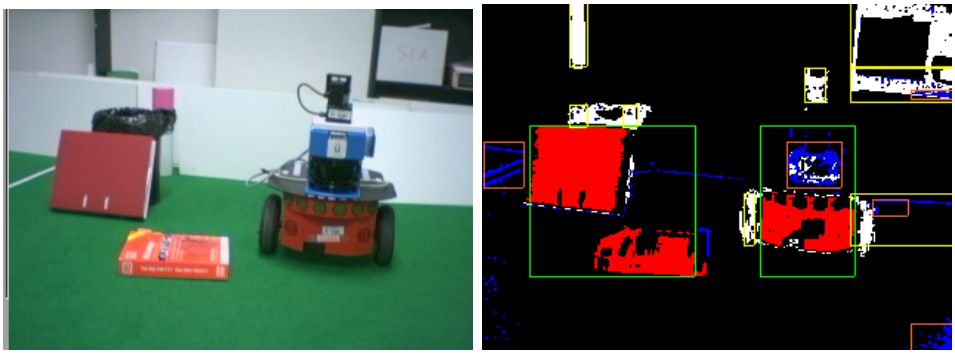
Figura 5.7: filtro RGB vs filtro HSI

Una vez obtenido un filtro de color bastante robusto, llevamos a cabo la segmentación de color basada en la imagen filtrada. La primera implementación fue la segmentación basada en la técnica del segundo umbral. Con está técnica se realiza una segmentación en la imagen, obteniéndose muy buenos resultados cuando se segmenta un color que es único en la imagen. Sin embargo cuando en la imagen hay varios objetos con el mismo color, está técnica no funciona tan bien. A modo de ejemplo podemos ver la figura 5.8(b), donde podemos apreciar que el histograma de las filas sólo tiene un segmento para el color rojo, de ahy que se segmente el libro y la carpeta como si fuera el mismo objeto.

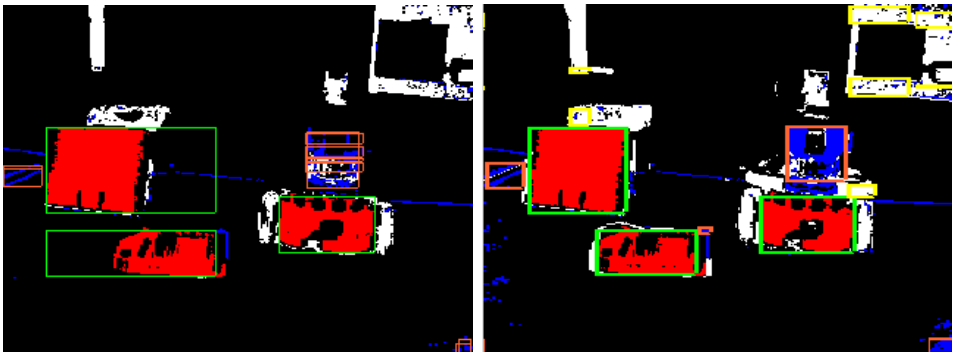
Al ver que con ésto no nos valía decidimos llevar a cabo una segunda segmentación sobre la primera, es decir una vez realizada la primera segmentación volvíamos a segmentar cada segmento. Como podemos apreciar en la figura 5.8(c), los resultados son mejores, aún así los segmentos de la carpeta y el libro no quedaban bien definidos, esta vez debido al histograma de las columnas. Además de que los segmentos no quedaran bien definidos, se segmentan todos los segmentos dos veces aunque queden bien definidos en la primera segmentación, por lo que perdíamos tiempo computacional.

Así implementamos un algoritmo donde solo se segmentaran los segmentos que no es-

taban muy bien definidos y todas las veces que fuera necesario. A este algoritmo lo denominamos: segmentación recursiva. Para ello cuando realizamos una segmentación, calculamos el porcentaje de pixeles de ese color con respecto al tamaño del segmento, si el porcentaje no supera un valor previamente determinado volveremos a segmentar dicho segmento. Como podemos ver en la figura 5.8(d) los resultados son excelentes y quedando los segmentos de cada objeto rojo bien definido.



(a) Imagen original (b) Primera segmentación



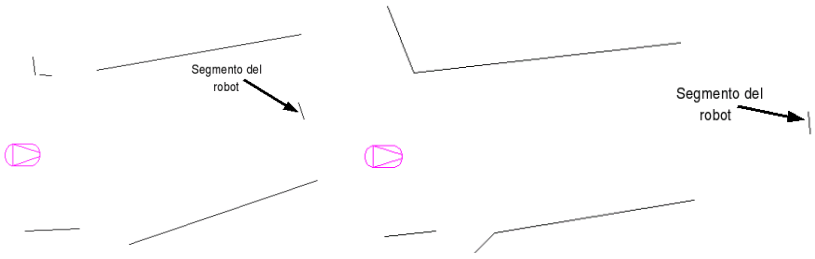
(c) Segunda segmentación (d) Segmentación recursiva

Figura 5.8: Segmentación recursiva vs primera y segunda segmentación

### 5.3.2. Subestímulo de profundidad

A parte de que el robot tenga que ser algo rojo, otra condición necesaria para que sea identificado es que el segmento de color rojo de sensación de profundidad. Como explicamos en el capítulo 4, para que haya de sensación de profundidad, el rayo visual entre el segmento rojo y la cámara tiene que cortar con un segmento del entorno, dado por el esquema *lasersegment*, y además éste no tiene que ser muy grande para poder evitar objetos parecidos que se encuentren en la pared, como pueda ser un poster.

Para extraer el subestímulo de profundidad del segmento rojo, probamos con dos implementaciones, donde los segmentos del entorno vienen dados por una lado por la memoria de puntos, y por otro lado por el sensor láser. Así hicimos una primera implementación donde los segmentos del entorno se basaban en los puntos del mapa local. Tras varias pruebas observamos que no se detectaba el segmento del robot hasta que éste estaba relativamente cerca, menos de dos metros, ya que al estar a más de un metro la densidad de los puntos del mapa local era menor, es decir la cantidad de puntos es menor por lo que éstos estaban más separados, así no se llegaba a crear ningún segmento. En la segunda implementación los segmentos del entorno se basaban en el sensor láser y pudimos comprobar que se detectava un segmento del robot a mucha más distancia. Como ejemplo podemos observar las figuras 5.9(b) y 5.9(a).




(a) Máxima distancia a la que se detecta el segmento del robot con la memoria de puntos

(b) Detección del robot a mayor distancia con el sensor láser

Figura 5.9: Extración del subestímulo de profundidad, (a) basandose en la memoria de puntos (b) basandose en el sensor láser

### 5.3.3. Suma heterogénea de subestímulos

Con el procesamiento de la imagen realizado y conseguido obtener el subestímulo de profundidad a una amplia distancia, sólo nos quedaba aplicar el *principio de suma heterogénea de estímulos* y dar los pesos correctos a cada uno de los subestímulos. Fue realmente fácil dar los pesos correctos, los valores de éstos son comentados en la sección 4.5.

Además conseguimos identificar al robot en diferentes distancias y posiciones: de lado donde se identifica al robot a una distancia máxima de tres metros (figura 5.10(a)), con

el láser visto frontalmente donde se llega a identificar al robot a una distancia máxima de tres metros y medio (figura 5.10(c)), y cuando el láser está detrás, es decir no se ve el láser, se llega a detectar al robot a una distancia máxima algo menor a los tres metros (figura 5.10(b).

(a) Robot identificado de lado

(b) Robot identificado viendo su parte trasera

(c) Robot identificado viendo su parte delantera

Figura 5.10: Identificación del robot con diferentes posiciones

Por último mostramos varios ejemplos donde teniendo objetos de color y tamaño parecido se detecta perfectamente al robot. En las siguientes figuras el robot se encuentra encuadrado en un rectángulo blanco y los candidatos en un rectángulo azul. En la figura 5.11(a) podemos ver como se identifica al robot teniendo al lado de una puerta roja, ya que el robot tiene los siguientes subestímulos, que la puerta no tiene,: sensación de profundidad, tiene algo azul y negro encima del segmento rojo. En la figura 5.11(b) vemos como se identifica al robot frente a una carpeta rojo, figura 5.11(c), en este caso ambos objetos tienen el subestímulo de profundidad pero el robot tiene los subestímulos de tener algo azul y negro encima del segmento rojo. Por último podemos ver como se identifica al robot, habiendo en la imagen un poster de éste a su lado. Aquí tanto el poster como el robot tienen todos los subestímulos, pero el poster no tiene el subestímulo necesario e imprescindible de tener sensación de profundidad ya que se encuentra en la pared.

(a) (b) (c)

Figura 5.11: Ejemplos de identificacion frente a objetos de tamaño y color parecidos

Una vez realizada las pruebas de identificación, podemos decir que nuestro algoritmo es suficientemente robusto y discriminante. Así detectamos al robot con situaciones en las que hay diferente iluminación. Además en las pruebas realizadas no hemos llegado a tener falsos negativos ni falsos positivos. También hemos conseguido identificar al robot en distintas posiciones: de lado, de frente viendo en la imagen el láser, viendo la parte trasera del robot donde no se ve el sensor láser. Varios ejemplos de lo robusto y discriminante de nuestro algoritmo lo podemos ver en la figura 5.11.

# Capítulo 6
# Conclusiones y trabajos futuros

Descrita la solución propuesta para este comportamiento y comentados algunos de los experimentos más relevantes, terminamos esta memoria exponiendo las conclusiones sacadas de este proyecto y las posibles líneas futuras en las que se puede seguir investigando.

## 6.1. Conclusiones

Haciendo un repaso de los objetivos marcados en el capítulo 2, vamos a describir en qué medida se han cumplido éstos satisfaciendo los requisitos también expuestos en ese capítulo. El objetivo principal se ha cumplido, y hemos conseguido un correcto comportamiento con todo el sistema integrado, como se ha validado en el capítulo 5.

El objetivo global estaba articulado en 3 etapas principales:

1. Desarrollar un algoritmo para que el robot deambule buscando al otro robot *Pioneer*, sin chocarse con ningún obstáculo.

2. Diseñar y desarrollar un método de identificación del robot, más robusto que el reconocimiento de patrones clásico de objetos.

3. Desarrollar la persecución del robot, y al igual que en el punto uno lo deberá hacer sin chocarse.

El primer subobjetivo era el de *buscar* al congénere por un entorno cerrado, sin chocarse con ningún obtáculo, ni estático ni dinámico. Para ello hemos programado y ajustado el algoritmo de navegación local *VFF* [J,Borenstein. 1989], para nuestro robot y configuración sensorial. Tal y como se vio en el capítulo 4 de la memoria se ha programado este algoritmo en un esquema, que se encarga de llevar al robot lejos de los obstáculos a la par que avanza hacia el destino, que en este caso será un punto

aleatorio en el centro del pasillo. Inicialmente probamos el algoritmo en el simulador *Player/Stage*. Una vez que comprobamos que el algoritmo funcionaba correctamente, sorteando incluso los objetos dinámicos que aparezcan, pasamos a probarlo en el robot real. En el robot real el algoritmo no funcionaba correctamente en determinadas ocasiones . Después ajustamos el comportamiento hasta que obtuvimos el comportamiento deseado. Como conclusión, destacamos la dificultad que tiene trabajar con el robot real. Una muy buena implementación teórica del comportamiento puede no funcionar correctamente sobre el robot real. Esto hace que las pruebas y experimentos lleven muchísimo tiempo.

El segundo subobjetivo era el de diseñar y desarrollar un método para identificar robustamente al otro robot. Como explicamos en el capítulo 4, nos basamos en un principio de la etología denominado: *suma heterógenea de estímulos*. Este principio dice que si la suma de un conjunto de subestímulos sencillo supera cierto umbral produce un comportamiento, en nuestro caso el congénere se asume identificado. Antes de aplicar este principio debemos procesar correctamente la imagen para poder sacar los subestímulos. Este procesamiento de la imagen (filtrado y segmentación de color) es una de las partes fundamentales, ya que si no se hiciera correctamente nunca se generaría de forma correcta el tercer subobjetivo. Para realizar un correcto procesamiento de la imagen tuvimos que realizar muchas pruebas, ya que en presencia en la imagen de objetos con el mismo color los resultados no eran los deseados.Una vez procesada la imagen correctamente, aplicamos el *Principio* anterior a los segmentos de color rojo , condición necesaria para ser el robot. Los subestímulos eran los siguientes: sensación de profundidad, segmento rojo más ancho que alto, que haya un segmento azul encima del segmento rojo y que haya un segmento negro encima del segmento rojo. Así si la suma de estos subestímulos, unos con mayor importancia que otros, nos dan la identificación o no del robot. Finalmente comprobamos que el algoritmo de detección funcionaba correctamente consiguiendo que fuera suficientemente discriminante y robusto. Además pudimos comprobar la gran dificultad de obtener información de las imágenes.

El útltimo subobjetivo era el seguimiento del robot *Pioneer*. Para llevar este último objetivo a cabo, y por tanto el comportamiento final, tuvimos que integrar antes los dos primeros subobjetivos. Para que la integración funcionara correctamente tuvimos que realizar algún pequeño ajuste. Una vez que dicha integración funcionaba correctamente, pasamos a realizar el seguimiento. El seguimiento es muy similar al primer subobjetivo ya que ambos usan la misma técnica de navegación local, pero con la di-

ferencia que los destinos, para el algoritmo de navegación local, son distintos.

Una vez verificado experimentalmente el comportamiento, podemos concluir que ha sido generado correctamente, y se ha desarrollado íntegramente en la plataforma *jde.c*, en la que como requisitos importantes debía implementarse el proyecto mediante esquemas, siendo programados en el lenguaje C, conseguir un algoritmo de navegación local vivaz y un algoritmo de identificación robusto.

## 6.2. Líneas futuras

En esta sección se detallan algunas posibles mejoras que se podrían realizar sobre este proyecto y que sirven como nuevas líneas de investigación para otros proyectos fin de carrera.

- Utilizar como entrada de nuestro mapa local información obtenida por dos cámaras, eliminando como entrada los valores obtenidos por el láser. Con esto conseguiríamos una representación en 3D del mundo real. Además detectariamos obstáculos no sólo en un plano como ocurre con e láser.
- Utilizar en nuestro comportamiento el movimiento del cuello mecánico, [Roberto Calvo, 2003]. Consiguiendo así que cuando el objetivo se pierda de vista , se pueda encontrar con suma facilidad, girando el cuello.
- Utilizar la corona delantera de sonar, para que no tengamos problemas con los cristales ya que el sensor láser no los detecta
- Insertar este comportamiento, en un robot con diferentes comportamientos, como por ejemplo que se pueda localizar. Llevar a cabo diferentes comportamientos, como puedan ser el de caza y huida dependiendo de la situación en la que se encuentre el robot.

[]

# Bibliografía


[ActivMedia, 2002] ActivMedia. Aria reference manual. *Technical Report version 1.1.10, ActiveMedia Robotics*, 2002.

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

[Borenstein, 1991] J. Borenstein. The vector field histogram - fast obstacle avoidance for mobile robots. *IEEE Journal of Robotics and Automation*, 1991.

[Brian P. Gerkey y Howard, 2003] Richard T. Vaughan Brian P. Gerkey y Andrew Howard. The player/stage project: Tools for multi-robot and distributed sensor systems. *Proceedings of the international conference on Advanced Robotics.*, pages 1–36, 2003.

[Calvo, 2004] Roberto Calvo. Comportamiento sigue persona con visión direccional. *Proyecto fin de carrera, URJC*, 2004.

[Díaz, 2005] Pedro Díaz. Navegación visual del robot pioneer. *Proyecto fin de carrera, URJC*, 2005.

[Gomez, 2002] Víctor Gomez. Comportamiento sigue pared en un robot con visión local. *Proyecto fin de carrera, URJC*, 2002.

[Gunter y Angermam, 1974] Vogel Gunter y Hartmat Angermam. Atlas de biologia. Editorial Omega, 1974.

[Isado, 2005] José Raul Isado. Navegación global por el método del gradiente. *Proyecto Fin de Carrera, URJC*, 2005.

[Lobato, 2003] David Lobato. Evitación de obstáculos basada en ventana dinámica. *Proyecto fin de carrera, URJC*, 2003.

[Lorenz, 1978] K. Lorenz. Fundamentos de la etología. Editorial Paidos, 1978.

[Martinez, 2003] Marta Martinez. Comportamiento sigue pelota con visión cenital. *Proyecto fin de carrera, URJC*, 2003.

[Martín, 2002] Félix San Martín. Comportamiento sigue pelota en un robot con visión local. *Proyecto fin de carrera, URJC*, 2002.

[Plaza, 2003] José María Cañas Plaza. *Jerarquía Dinámica de Esquemas para la generación de comportamiento autónomo*. PhD thesis, Universidad Politécnica de Madrid, 2003.

[Plaza, 2004] José María Cañas Plaza. Manual de programación de robots con jde. *URJC*, pages 1–36, 2004.

[R.Calvo y Pérez, 2005] J.M.Cañas R.Calvo y L.García Pérez. Person following behavior generated with jde schema hierarchy. *Conf. on Informatics in Control, Automation and Robotics*, 2005.