UNIVERSIDAD REY JUAN CARLOS

Ingeniería Técnica en Informática de Sistemas

Escuela Superior de Ciencias Experimentales y Tecnología

Curso académico 2003-2004

Proyecto Fin de Carrera

Comportamiento sigue persona con visión.

Tutor: José M. Cañas Plaza

Autor: Ricardo Ortiz Herencia

Junio 2.004

Dedicado a las víctimas de los atentados terroristas sucedidos el once de Marzo en Madrid.

Agradecimentos.

Gracias a Jose M. Cañas Plaza y a Vicente Matellán Olivera por el tiempo y dedicación puesto en mí y en este proyecto. Así como a todos y cada uno de los miembros del Grupo de Robótica de la URJC por su gran colaboración.

Gracias a mis padres por tantos años de buenos consejos.

Y gracias a Jose Luis Hidalgo por diez años de dedicación a ese gran proyecto llamado amistad.

Índice general

Índice de figuras

Índice de cuadros

Resumen

Algunos de los últimos avances realizados en el campo de la robótica han ido enfocados a mejorar la interacción entre persona y robot. Un comportamiento autónomo interesante en esta línea de investigación es el de seguimiento.

El comportamiento desarrollado en este proyecto consigue que un robot móvil siga a una persona, vestida con ropa llamativa, a la vez que evita los posibles obstáculos que puedan aparecer en su camino, y manteniéndo una distancia prudencial con la persona. Para ello el robot se ayuda de una cámara para localizar a la persona, aplicando un filtrado de color sobre la imagen, con lo que separa a la persona del resto de objetos que capta la cámara. En función de la dirección en la que se encuentre la persona, envía órdenes de movimiento a los motores del robot para dirigirse hacia ella, comprobando con las medidas recogidas por los sensores que si se mueve en esa dirección no va a chocar contra algún obstáculo. Si es necesario evitar algún obstáculo, lo hace tratando de no perder de vista a la persona. Si ésto último sucediera, el robot la busca, siguiendo una dirección aproximada a la que llevaba la última vez que la localizó en la imagen.

Este comportamiento se ha materializado como un *control reactivo basado en visión*, dentro de la plataforma software JDE. Esta arquitectura permite la creación de *"esquemas"* concurrentes, y en concreto para el comportamiento sigue persona se crearon dos: uno de ellos encargado de la parte de visión (*esquema perceptivo*) y otro encargado de la parte de actuación(*esquema motor*). La ejecución concurrente de ambos esquemas permite generar el comportamiento. Lo que analiza el esquema perceptivo es aprovechado por el esquema de actuación para enviar una orden de movimiento al robot. Se han realizado diferentes experimentos en varios escenarios de interiores con el robot Pioneer. En las pruebas realizadas se ha validado experimentalmente el diseño y la implementación desarrollados. Todos los programas necesarios para llevar a cabo este comportamiento se implementaron con el lenguaje de programación C, y sobre el sistema operativo GNU/Linux.

Capítulo 1

Introducción.

El contexto en el que se encuadra este proyecto es la robótica, con lo que sería de gran ayuda tomar como punto de partida su definición. Se admite que robótica es la ciencia de los robots. El concepto de robot tiene su origen en 1921, con la obra de teatro "RUR"(*Rossum's Universal Robota*), del checo Karel Capek, de donde surge la palabra robot, ya que *robota* en checo significa esclavo. Por lo general el concepto que se tiene de robot es el de dispositivo humanoide, con un cierto grado de inteligencia, que reemplaza a los humanos en la realización de tareas útiles. Esta idea la han creado en parte películas como *"La Guerra de las Galaxias"*, en la que aparecen robots que se adaptan en mayor o menor medida a esa definición, creándose una relación equivocada que asocia robot de ciencia ficción con robot real.

El concepto de robot entró en otra fase entre los años 70 y 80 con los robots industriales, en la que hablar de robot era hablar de manipuladores usados en plantas industriales para desempeñar una tarea específica y aislados a su vez de nuestro entorno como medida de seguridad. Un ejemplo lo tenemos en los brazos soldadores o pintores usados en las fábricas de coches. Este tipo de robots surgen como medida para realizar procesos de producción mucho más eficientes y mejorar la calidad de los productos. Pero en los años 80 y principios de los 90 surge un nuevo enfoque en la robótica y con ello, un nueva definición de robot. Este tipo de robótica se conoce como robótica móvil y pone mayor énfasis en la autonomía en entornos dinámicos no controlados.

Los robots móviles autónomos son sistemas completos que operan eficientemente en entornos complejos sin necesidad de estar constantemente guiados y controlados por operadores humanos. Una propiedad fundamental de los robots autónomos es la de poder realizar por sí mismos distintas tareas según las características del entorno en un momento dado. La funcionalidad de los robots autónomos es muy amplia y variada, desde robots que cumplen un servicio determinado hasta robots usados para el entretenimiento.

Un ejemplo claro de esta clase de robótica lo tenemos en la figura 1.1, donde podemos apreciar en la derecha al robot Asimo, creado por Honda[1]. Un robot cuya

[1] `http://world.honda.com/ASIMO/`

funcionalidad va encaminada al entretenimiento es el perro Aibo de Sony, que trata de simular todos los movimientos y emociones propias de un animal doméstico[2]. Existen a su vez robots dedicados a cumplir un determinado servicio como la ayuda a personas discapacitadas, como pueden ser Pearl, del proyecto Nursebot, de las universidades de Carnegie Mellon y de Pittsburgh (EE.UU.)[3], que pertenece a la gama de "robots enfermeros" encargados de recordar a un enfermo cuando ha de tomarse un determinado medicamento o analizar su ritmo cardiaco.

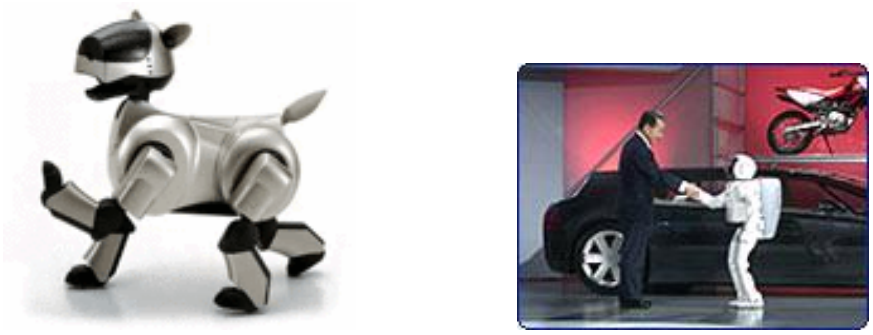

Figura 1.1: Perro Aibo y robot bípedo Asimo.

En cualquiera de estos casos se tiende a crear robots que interactúen con las personas. Si la robótica industrial está encaminada a mejorar la producción en las diferentes fábricas, la robótica móvil autónoma dirige sus últimos avances a lograr un interfaz hombre-máquina. Y es en estos avances donde se encuadra, a grandes rasgos, nuestro comportamiento sigue persona con visión.

1.1. Grupo de Robótica de la URJC

El grupo de robótica de la URJC[4], formado por profesores y alumnos, trabaja con temas relacionados con la robótica y la Inteligencia Artificial aplicada(IA). En la actualidad el grupo dispone de dos robots Pioneer, dos perritos Aibo de Sony, 6 robots EyeBot[5] y varios robots Lego.

Las líneas de investigación desarrolladas en el grupo van enfocadas a dos tipos de trabajos. Uno de ellos es la creación de un equipo de fútbol para competir en la RoboCup[6] y para ello se han desarrollado proyectos sobre el robot EyeBot. Entre estos proyectos se encuentran algunos comportamientos de seguimiento como el *Comportamiento sigue pelota con visión cenital* [Martinez03], *Comportamiento sigue pared* [Gomez02] o el *Comportamiento sigue pelota en un robot con visión local* [SanMartín02].

[2] `http://www.sony.net/Products/aibo/`
[3] `http://www-2.cs.cmu.edu/ñursebot/`
[4] `http://gsyc.escet.urjc.es/robotica/`
[5] `http://www.ee.uwa.edu.au/ braunl/eyebot/`
[6] `http://www.robocup.org/`

La segunda línea de investigación se centra en crear comportamientos autónomos en entornos de interiores, como laboratorios y oficinas. Para la creación de estos comportamientos autónomos se diseñó la arquitectura JDE (Jerarquía Dinámica de Esquemas) [Cañas03]. Ésta es una arquitectura de control que basa su funcionamiento en esquemas que pueden comunicarse entre sí, dividiéndose en esquemas perceptivos y motores. Los esquemas perceptivos recogen información sensorial, que más tarde será aprovechada por un esquema motor para envíar una órden de movimiento al robot. De modo que si un esquema motor necesita información de un determinado esquema perceptivo lo activa. Este esquema perceptivo a su vez puede activar otros esquemas perceptivos para obtener esa información, formándose así una jerarquía construida dinámicamente entre los diferentes esquemas.

En la actualidad, el grupo trabaja en la creación de comportamientos autónomos alrededor del robot Pioneer, bajo la plataforma de JDE. Uno de estos comportamientos es el comportamiento sigue persona con visión, el cual se describe a continuación. Este comportamiento, junto con el *Comportamiento sigue persona con visión direccional* [Calvo04], son los primeros comportamientos autónomos implementados bajo el robot Pioneer en el grupo que abordan una interacción con la persona. Anteriormente se había diseñado un comportamiento basado en una navegación local segura como la *evitación de obstáculos basada en ventana dinámica* [Lobato03] sobre este mismo robot.

1.2. Comportamiento sigue Persona con Visión.

Siguiendo la línea de investigación del grupo a la hora de crear comportamientos de seguimiento, el presente proyecto está enfocado a generar un comportamiento autónomo de seguimiento sobre un robot real. Existen cada vez más robots que entran en los hogares para desempeñar una función de ayuda al hombre. Ésto implica que el robot debe realizar un seguimiento de la persona por esta clase de escenarios. Por eso el generar comportamientos autónomos de seguimiento se muestra como una gran utilidad para mejorar la interacción hombre-máquina.

Este comportamiento es un caso particular de *control reactivo guiado por visión*, cuya implementación genera un sistema que puede ejecutarse en entornos reales, estableciendo de alguna manera una conexión inteligente entre percepción y acción. El comportamiento consiste en realizar el seguimiento de una persona, a través de visión, reaccionándo de manera efectiva antes las diferentes situaciones que puedan producirse (evitación de obstáculos, pérdida de la persona..).

El robot sigue a la persona por entornos de interiores, como pasillos o habitaciones. A su vez, la persona irá vestida con una camiseta de color llamativo con el fin de facilitarle su localización al robot. Al mencionar que este comportamiento es un control reactivo basado en visión, nos referimos a que no existe planificación clásica de

trayectorias, es decir, el robot actúa según la información que recoge de sus sensores. Con respecto a otro proyecto realizado también sobre el Pioneer, como la *evitación de obstáculos basada en ventana dinámica* [Lobato03], el comportamiento sigue persona se diferencia en que no hay dirección objetivo fija, sino que esta dirección varia según donde se encuentre la persona que se está siguiendo. Existe también alguna diferencia con respecto a otro proyecto realizado en la actualizadad sobre el Pioneer, como es el *comportamiento sigue persona con visión direccional* [Calvo04]. Este comportamiento incorpora un cuello mecánico para orientar la cámara en diferentes direcciones sin necesidad de mover el robot. En nuestro comportamiento, la orientación de la cámara es fija moviéndose solidaria al robot.

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 analizan con detalle el robot sobre el que se realizaron los experimentos, y los diferentes dispositivos de los que dispone. En la parte software hablaremos de la plataforma de programación sobre la que se ha desarrollado el comportamiento, llamada JDE. Seguidamente encontramos 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. Desde que el robot localiza a la persona, hasta que se mueve en esa dirección. Para finalizar este capítulo se describen una serie de experimentos realizados sobre el robot real. Finalmente se describen las conclusiones obtenidas y un línea de posibles trabajos futuros sobre el mismo proyecto.

Capítulo 2

Objetivos.

Una vez presentado el contexto de este proyecto, a continuación se describen los objetivos fundamentales, además de los requisitos necesarios que nuestra solución ha de tener en cuenta. En un principio el objetivo fundamental es que un robot sea capaz de seguir a la persona. Se plantea cierto nivel de complejidad a la hora de generar este comportamiento, ya que aparecen problemas y dificultades que hay que resolver si se quiere que el robot posea cierta autonomía.

2.1. Objetivos

El objetivo global de este proyecto es generar el comportamiento de seguimiento en un robot real. Este objetivo se compone de tres subobjetivos: el primero de ellos se centra en el diseño, ya que se ha de diseñar el comportamiento dentro del modelo cognitivo de JDE. El segundo se centra en la implementación, ya que se han de desarrollar los programas necesarios para materializar el comportamiento bajo la base conceptual de JDE. Y el tercero se refiere a los experimentos, ya que el comportamiento ha de funcionar correctamente en un robot real.

Dentro del comportamiento de seguimiento han de combinarse dos tendencias, la primera se refiere al seguimiento de la persona, ya que el Pioneer deberá de realizar éste de la manera más natural posible y en tiempo real, sin efectuar cambios bruscos en su velocidad y manteniendo a la persona a una cierta distancia. La segunda se centra en la evitación de obstáculos, ya que el seguimiento ha de llevarse a cabo en cualquier entorno sin chocar contra ellos. Este último objetivo debe ser, por supuesto, seguro y fiable, asegurándonos que el robot va a ser capaz de no chocar con nada ni con nadie mientras efectúa el seguimiento, estableciéndose así un compromiso *seguridad-seguimiento* entre ambos subobjetivos.

En lo relacionado al seguimiento se encuadran varios aspectos que merece la pena destacar. En el proceso de seguimiento el robot ha de mantenerse alineado con la persona, de tal manera que si ésta se desplaza hacia la derecha el robot ha de hacerlo

también en esa dirección. También ha de tener en cuenta que si mientras realiza el seguimiento la persona desaparece de su campo visual, debe iniciar una búsqueda hasta que vuelva a encontrarla.

Ha de mantener una cierta distancia con respecto a la persona, de modo que si percibe que ésta se encuentra muy cerca se pare, y continúe siguiéndola cuando se aleje. En lo relacionado a la velocidad, el robot durante el seguimiento no debe ir a tirones, es decir, si tiene que acelerar porque la persona se aleja, lo ha de hacer de manera gradual y con suavidad, y nunca de manera brusca. La velocidad durante el seguimiento ha de permitir al robot seguir a una persona que camina con paso normal por diferentes entornos (pasillos, habitaciones..).

Dentro del compromiso asociado a la seguridad se encuadran también otros objetivos, uno de ellos se refiere a la evitación de obstáculos, ya que el robot debe ser capaz de percibirlos para posteriormente evitarlos. Ha de poder controlar la velocidad que lleva en todo momento dependiendo de si alrededor tiene muchos obstáculos, en cuyo caso no irá muy rápido, o por el contrario, en el entorno que le rodea los obstáculos están suficientemente lejos del robot para que pueda alcanzar una velocidad mayor a la hora de seguir a la persona. A su vez, si el obstáculo no está demasiado cerca del robot, la trayectoria que estaba siguiendo hasta ese momento no ha de desviarse tanto como cuando tiene el obstáculo muy cerca. Ésto último ha de cumplirse ya que se trata de un comportamiento sigue persona y no un comportamiento de evitación de obstáculos, con lo cual, siempre que sea posible, se debe de evitar al obstáculo pero sin desviarse demasiado de la dirección en la que se encuentra la persona.

2.2. Requisitos

A continuación se detallan los requisitos fundamentales que ha de cumplir el comportamiento sigue persona, algunos de ellos situados en la parte hardware y otros situados en la parte software.

1. Robot Pioneer

   El primer requisito necesario para llevar a cabo este comportamiento se encuentra en el robot Pioneer, ya que nuestra aplicación está diseñada para ejecutarse bajo este tipo de robot, teniendo en cuenta su estructura y los dispositivos sensoriales que posee.

2. Arquitectura JDE

   La arquitectura JDE es la forma que tenemos de combinar lecturas para llevar a cabo una decisión de actuación. Simplifica enormemente la complejidad de la

aplicación, pues resuelve aspectos como la captura de imágenes, la interfaz gráfica o el envío de comandos a los motores.

El uso de esta arquitectura hace mucho mas sencillo abordar el problema no sólo del comportamiento sigue persona con visión, sino cualquier otro comportamiento autónomo.

3. Vivacidad

   La vivacidad es requisito importante a la hora de conseguir que el robot no sólo siga a la persona, sino que lo haga de manera dinámica. El movimiento ha de ser continuo y los cambios de velocidad han de hacerse de forma progresiva, de tal forma que el robot sólo se detenga cuando tenga a la persona muy cerca, o pasado un tiempo que no la visualiza en la imagen.

   Esto se consigue si el comportamiento tiene un alto grado de vivacidad, es decir, si se envía una orden de movimiento a los motores del robot, esta orden se ejecuta de manera inmediata. Para ello es necesario poder enviar varias órdenes de movimiento al robot por segundo, ya que se trata de un control reactivo basado en visión. Se observa en otros comportamientos, como el *comportamiento sigue pared* [Gomez02] o el *comportamiento sigue persona con visión cenital* [Martinez03], que cuanto mayor es el número de órdenes de movimiento que se le envían al robot por segundo, mejor es la calidad del comportamiento.

   Esto impone la restricción de que los algoritmos que se usen han de ser computacionalmente livianos, es decir, no han de consumir mucha CPU.

4. Robustez

   En el apartado de percepción, el robot ha de ser capaz de identificar a la persona en la imagen. Para ello, la persona deberá ir vestida con algún tipo de ropa llamativa que la distinga del resto de objetos y colores que el robot capta a través de su cámara. El uso de una ropa llamativa en el comportamiento se debe a que tratamos de ofrecer facilidades al robot perceptivamente, y dar más importancia al seguimiento. Ha de cumplir por tanto una cierta robustez en la percepción, de forma que pueda localizar a la persona en zonas que sufran fuertes cambios de iluminación.

Capítulo 3

Herramientas

En este capítulo se describe, en primer lugar, al robot sobre el que se ha realizado este comportamiento y los dispositivos de los que éste dispone, algunos de ellos empleados en este proyecto como el sensor láser de proximidad, ultrasonidos o la cámara. En segundo lugar se analizarán las principales herramientas software que sirvieron para analizar y organizar toda la información procedente de esos dispositivos, comenzando por la arquitectura JDE, plataforma software sobre la que se desarrollan los programas que generan nuestro comportamiento.

3.1. Robot Pioneer

Dentro de la gama de robots del fabricante ActivMedia[1], se encuentra la serie de robots Pioneer con los que actualmente trabaja el Grupo de Robótica de la URJC ([Cañas03b]). En concreto se trata de el Pioneer 2DXE (figura 3.1), un robot móvil dotado de dos ruedas laterales y una tercera rueda loca, conformando un diseño en triángulo. El robot tiene como actuadores a dos motores de continua cada uno asociado a una rueda lateral. Dispone además de una serie de sensores, tanto internos como externos, que sirven para recoger información no sólo sobre el entorno que rodea al robot, sino también sobre su velocidad actual y posición.

Dentro de estos sensores se encuentran dos odómetros(*encoders*) asociados a cada una de las ruedas laterales y una serie de sensores externos, entre los que se encuentran los tactiles situados al frente y en la parte trasera del robot, y un anillo de 16 ultrasonidos. Después se le han añadido otros dos sensores externos más, uno de ellos es el sensor láser de proximidad y en la parte de visión, una cámara digital. La utilidad de estos sensores así como su aportación a este comportamiento se tratan más adelante.

La lectura de toda la información relativa a los ultrasonidos, los odómetros y los tactiles se lleva a cabo en un sistema operativo llamado P2OS que se ejecuta bajo un microcontrolador Siemens 88C166 a 20 MHz situado en el interior del robot. Este sistema operativo es el encargado de llevar a cabo las órdenes de movimiento que le

[1] http://www.activmedia.com/

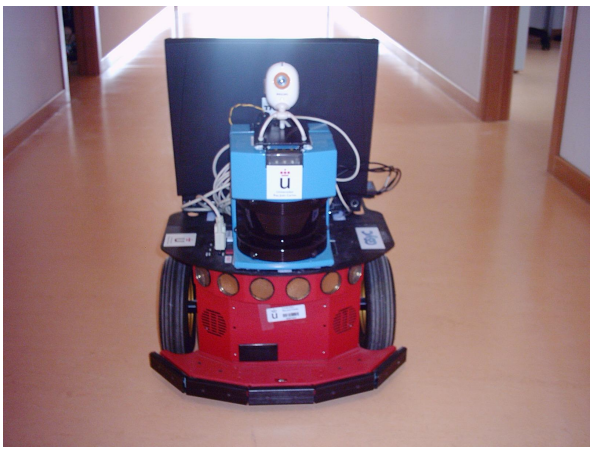

Figura 3.1: Robot Pioneer

van llegando sobre los motores. Tanto la lectura de datos sensoriales como el envío de órdenes a los motores se efectúa a través de tarjetas especiales.

La base del Pioneer está conectada a través de puerto serie a un PC, en concreto un portátil con procesador Pentium III a 600 MHz, situado sobre el robot y sobre el que se ejecutan los programas de alto nivel necesarios para llevar a cabo éste y otros comportamientos. Es precisamente a través de este puerto serie RS232 por el cual el PC recoge las lecturas de odómetros y sónares procedentes del P2OS y envía órdenes de movimiento al mismo para que las lleve a cabo sobre los motores. Los otros dos dispositivos mencionados anteriormente, láser y cámara, se conectan al portátil mediante puerto serie y USB respectivamente. Es posible conectar este ordenador a otro cualquiera a través de red inalámbrica gracias a una tarjeta de red 802.11, capaz de alcanzar una velocidad de 11 MBps. Esta herramienta resulta muy útil si queremos analizar las medidas y gobernar al robot desde cualquier otra máquina distinta al PC portátil.

Toda esta estructura hardware descrita se visualiza en el diagrama de bloques de la figura 3.2, así como la distribución de los diferentes componentes en la figura 3.3.

3.1.1. Sensores Sonar

Este sensor, sin ser tan preciso como el láser, muestra una valiosa información sobre lo que rodea al robot, y no sólo de su parte frontal como en el caso del láser, sino que támbien da información sobre la parte trasera del robot. El robot Pioneer posee un anillo de 16 sónares, ocho frontales y ocho traseros. Su funcionamiento se basa en la emisión de ondas de ultrasonido que rebotan en los obstáculos o superficies formando el eco, que regresa pasado un tiempo al punto de emisión. Es entonces cuando se calcula el tiempo de ida y vuelta conocido como *tiempo de vuelo*. A partir de ahí y conociendo la velocidad del ultrasonido se calcula la distancia recorrida por la onda, que será justo el doble de la distancia al obstáculo. Esta onda se propapa de forma circular llegando

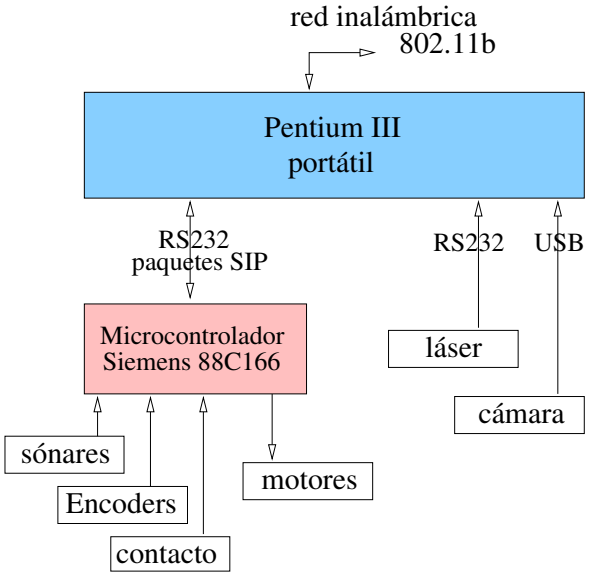

Figura 3.2: Diagrama de bloques Hardware

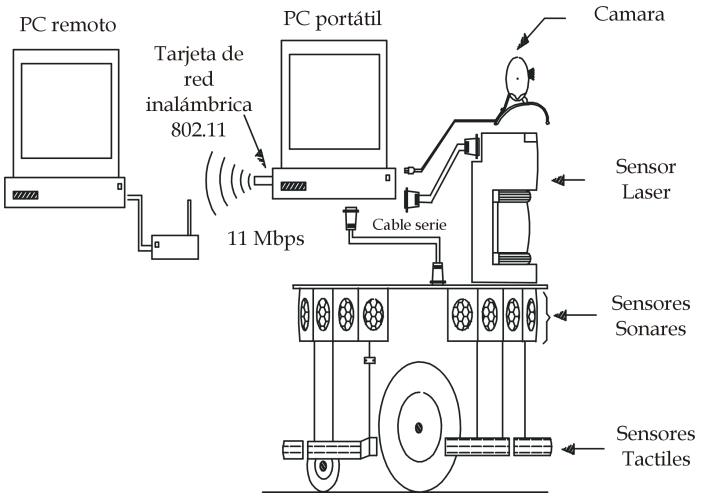

Figura 3.3: Distribución hardware de dispositivos

a formar un cono de 20 grados como puede verse en la figura 3.5. En el caso concreto del robot Pioneer, la máxima distancia medida es de tres metros.

Esta clase de sensor tiene varias incertidumbres en sus medidas. La primera de ellas es la *incertidumbre angular* que se produce cuando un obstáculo se encuentra dentro del cono de la onda de propagación, ya que se puede calcular la distancia pero la posición real del obstáculo es desconocida. Ésto es debido a que podría encontrarse a cualquier distancia d con respecto al origen dentro del cono. En segundo lugar, este tipo de sensores también pueden tener *reflexiones especulares* provocadas por diferentes superficies (las superficies lisas provocan que las ondas reboten en diferentes direcciones, y tras múltiples rebotes llegan al sensor, que entonces sobreestima la distancia al obstáculo) y por distintos ángulos de incidencia, ya que la dirección del reflejo depende de este ángulo. En tercer lugar, el *error radial* es otro problema que poseen esta clase

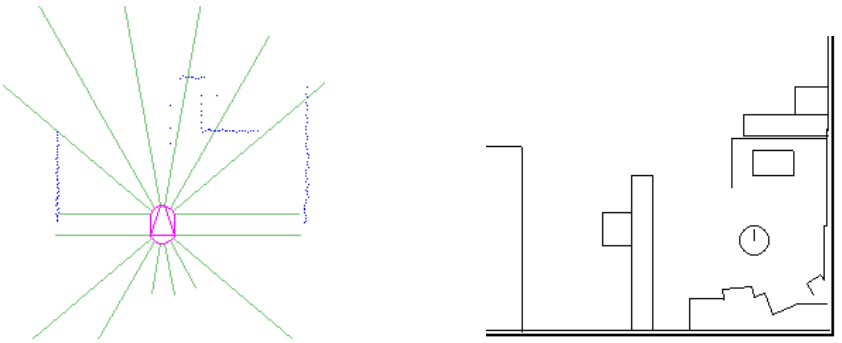

Figura 3.4: Distribución de los sensores de ultrasonidos en el Pioneer.

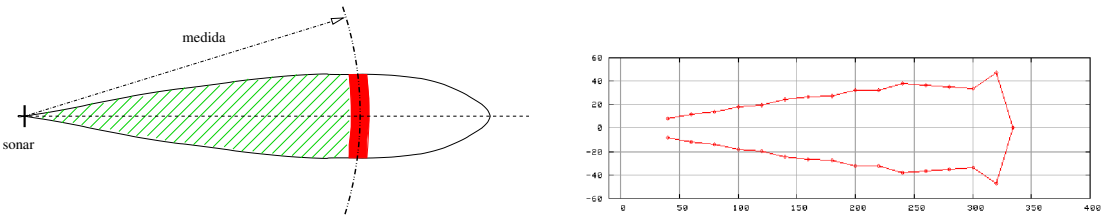

Figura 3.5: Interpretación de la lectura sonar (izquierda) y alcance experimental (derecha).

de sensores y se produce cuando se calcula una estimación inexacta de la distancia al obstáculo (por variaciones de la velocidad real del ultrasonido, por la temperatura..). En la figura 3.4 se muestra el radio de acción de los sónares(verde).

3.1.2. Sensor láser

El sensor láser de proximidad permite al robot tener un conocimiento muy preciso de su entorno, y por tanto, una herramienta muy útil para la detección de obstáculos. El sensor concreto utilizado fue el modelo LMS-200 de SICK[2]. Este sensor traza una semicircunferencia de 180 grados de amplitud y tiene una profundidad máxima de ocho metros. Dentro de ese radio de acción, el láser efectúa diez barridos por segundo con una precisión de 1 grados y 2cm, lecturas que el ordenador portátil recoge a través del puerto serie. Este sensor no ofrece información sobre la totalidad del entorno que rodea al robot, ya que sólo lo hace de su parte frontal, aun así, su precisión sobre esta zona es superior a la que puedan ofrecer otro tipo de sensores como son los sensores de ultrasonido. En la figura 3.6 vemos un ejemplo de una lectura láser para un mundo en concreto.

3.1.3. Cámara

Otro sensor usado como herramienta de visión en el comportamiento ha sido la cámara digital de la serie PCVC-740K de Philips (3.7). Se conecta al PC directamente a través de puerto USB y sin necesidad de tarjeta digitalizadora, siendo ésto último

[2] http://www.sick.de/de/en.html/

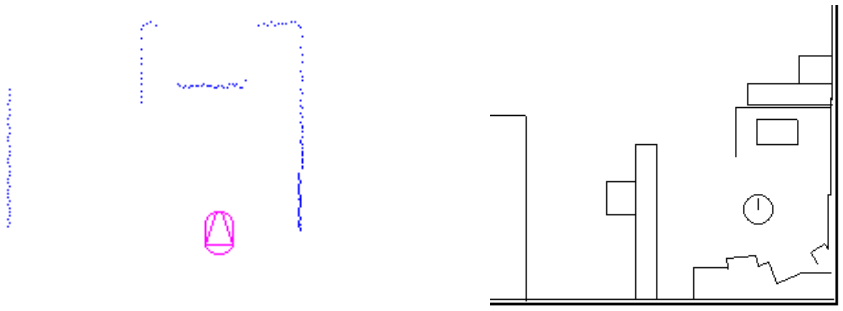

Figura 3.6: Radio de acción del sensor láser.

lo que la diferencia de las cámaras analógicas. Entre sus principales características cabe mencionar su ángulo de visión de 60 grados de amplitud en horizontal, como puede verse en la figura 3.8, y una buena velocidad de captura (de hasta 25 fps) a una resolución de 320x240. Incorpora un sensor CCD de alta resolución, lo que le permite tomar imágenes con luminosidad inferior a 1 lux. Al proporcionar imágenes en color, supone una herramienta muy importante a la hora de localizar e identificar a la persona para realizar el posterior seguimiento.

La manera de capturar imágenes se realiza gracias a la interfaz `Video4Linux`. Esta interfaz da soporte a diferentes capturadoras de video y televisión, y a cámaras USB. En concreto, `Video4Linux` incluye al driver `pwc`[3] para dar soporte a diferentes cámaras de la casa Philips. Además del `pwc`, existe otro driver llamado `pwcx` que envía imágenes comprimidas por USB, y gracias a eso podemos obtener velocidades de captura de hasta 25 fps con una resolución de 320x240..

Figura 3.7: Cámara Philips PCVC740K

3.1.4. Odómetros y Tactiles

El Pioneer dispone de otros dos tipos de sensores como son los odómetros y los sensores tactiles. En el caso de los odómetros, asociados a cada una de las ruedas, su

[3] http://www.smcc.demon.nl/webcam

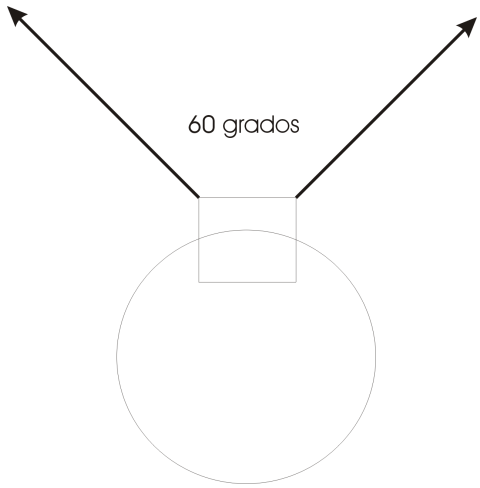

Figura 3.8: Posición y ángulo de visión en horizontal de la cámara

funcionamiento se basa en contar los pulsos que se generan a medida que gira el eje. La cuenta asciende hasta 500 pulsos por cada giro completo de la rueda, lo que se traduce en 66 ticks/mm. Su uso principal es para estimar la posición en la que se encuentra el robot, aunque también sirven para estimar la velocidad real a la que se mueve. Su uso no fue necesario en el desarrollo de este comportamiento ya que no se lleva a cabo un control en posición. En sus estimaciones estos sensores acumulan un error debido al deslizamiento de las ruedas, y a los desalineamientos.

El conjunto de sensores tactiles se compone de 5 delanteros y 5 traseros en el caso del robot Pioneer, de tal forma, que si algún obstáculo impacta contra alguno de ellos, el robot ofrecerá un lectura binaria de choque o no choque. Su uso pues, permite detectar obstáculos, aunque en el caso del comportamiento sigue persona se considera como último recurso, ya que a priori se trata de detectar obstáculos sin chocar contra ellos.

3.1.5. Motores

Para terminar con la descripción del hardware del Pioneer, hablaremos de los actuadores que éste posee. El robot Pioneer dispone de dos actuadores que corresponden a los dos motores de continua, cada uno de ellos asociado a una rueda. Con estos actuadores el robot logra un movimiento de tracción diferencial (tipo tanque), lo que le permite girar sobre sí mismo, girando una rueda en un sentido y la otra en el contrario, ir recto, girando las dos ruedas a la misma velocidad, y trazar curvas, girando ambas ruedas en el mismo sentido a diferente velocidad. De esta forma, el robot puede alcanzar velocidades de hasta 1,8 m/s en su velocidad lineal y 360 grados/s en su velocidad angular.

3.2. JDE

La plataforma sotware sobre la que se ha diseñado nuestro comportamiento es JDE (Jerarquía Dinámica de Esquemas) [Cañas03][Cañas04]. Esta arquitectura ha sido diseñada por el Grupo de robótica de la URJC [4] y tiene como unidad de comportamiento a los denominados *"esquemas"*, cada uno encargado de realizar una tarea específica. En el caso del comportamiento sigue persona con visión, JDE se muestra como una herramienta útil en dos puntos fundamentales: Como plataforma software en el que se integra el código que genera nuestro comportamiento, y como referencia conceptual cognitiva para organizarlo.

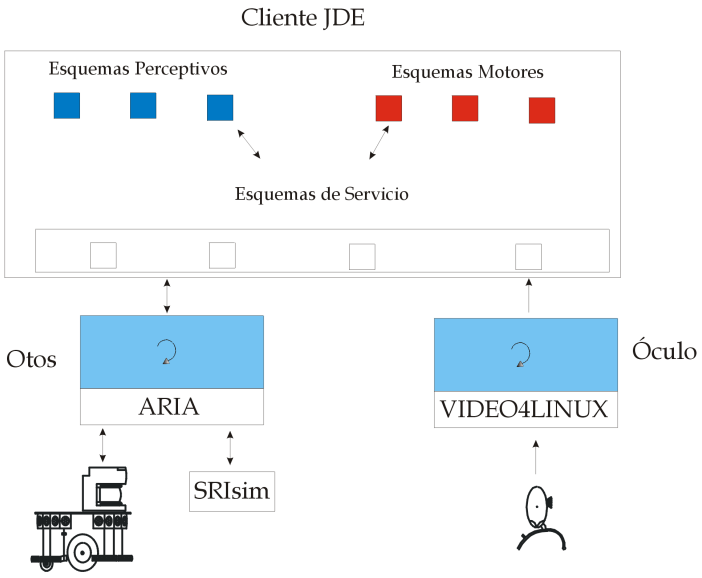

Figura 3.9: Arquitectura Software de JDE

3.2.1. Arquitectura Cognitiva

JDE se basa en un tipo de arquitectura de control llamados *Sistemas Basados en Comportamientos* (SBC). Este tipo de arquitectura poseen un enfoque "reactivo", es decir, actúan según su interacción con el entorno, distribuyendo el control y la percepción entre diferentes unidades de comportamiento. En el caso de JDE, la unidad de comportamiento se denomina *"esquema"*, y se organizan en jerarquías. Un esquema es un hilo de ejecución implementado de forma iterativa y encargado de una tarea específica. Estos esquemas pueden ejecutarse simultáneamente, activarse o desactivarse, modularse e interactuar entre sí.

Los esquemas se dividen en *perceptivos* y *motores*, siendo los perceptivos los encargados de leer datos sensoriales y poner esa información a disposición de cualquier

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

esquema. Los esquemas motores utilizan la información recogida por los esquemas perceptivos para generar una orden de movimiento sobre el robot.

Como su propio nombre indica, JDE organiza sus esquemas en una jerarquía dinámica, de modo que si un esquema motor requiere cierta información sensorial, activa el esquema perceptivo encargado de actualizar esa información, a su vez, este esquema perceptivo podrá activar otros esquemas perceptivos para obtener esa información, creándose así una jerarquía dinámica. Un esquema que activa a otro porque necesita de sus servicios también puede modularlo para que se ejecute a un determinado ritmo, dependiendo así el funcionamiento del esquema activado de quién lo active, y cómo lo module. En el caso del comportamiento sigue persona no hemos necesitado de los mecanismos jerárquicos, ya que nos ha bastado con un único nivel.

La manera que tienen estos hilos de ejecución de comunicarse entre ellos es a través de *memoria compartida*, de modo que si un esquema perceptivo lee un determinado dato sensorial, actualiza una variable común y el resto de esquemas leen esta variable cuando la necesiten. Cada vez que se habla de comunicación entre procesos a través de memoria compartida, se habla de condiciones de carrera. En la arquitectura JDE, se producen condiciones de carrera en el acceso a las variables compartidas, pero no se implementan semáforos con el fin de evitar sobrecarga computacional. Estas condiciones de carrera pueden provocar que un esquema motriz envíe una orden errónea de actuación en una determinada iteración, pero al enviar varias de estas órdenes por segundo, este error se puede corregir en las sucesivas iteraciones, sin que en el movimiento del robot se note la diferencia. De modo que no es imprescindible el uso de semáforos en este caso, y su utilización supondría un mayor coste computacional en la plataforma.

Es en este nivel de la arquitectura JDE donde se integra los programas creados para generar el comportamiento sigue persona con visión, de modo que podamos separar la parte de percepción de la parte de acción.

3.2.2. Servidores y Clientes

Hemos hablado de los dos tipos de esquemas, perceptivos y motores, que pueden existir en el nivel alto de la arquitectura JDE, adelantando que los programas creados para generar el comportamiento sigue persona se integrarán dentro de este tipo de esquemas. En este mismo nivel, la colección de esquemas JDE asume el papel de cliente que se conecta a dos servidores, `otos` y `óculo`, situados un nivel por debajo de él. Estos servidores cumplen la función de poner en disposición de los clientes las medidas sensoriales del robot, así como la posibilidad de enviar órdenes de movimiento.

El servidor `otos` es el servidor encargado de dar acceso a los sensores láser, ultrasonidos, odómetros y tactiles, además de ofrecer acceso a los motores. Esta

comunicación entre el cliente y el servidor se realiza a través de una interfaz de mensajes como parte de un protocolo desarrollado exclusivamente para esta arquitectura, de tal manera, que si por ejemplo un cliente necesitara conocer las medidas del sensor láser, deberá enviar un mensaje a `otos` indicándole que quiere suscribirse a tal sensor mediante *OTOS_subscribe_laser*. Una vez se ha establecido una comunicación entre un cliente y `otos`, y se ha realizado la subscripción al sensor, el envío de información entre ambos no tiene porque ser continuo, ya que `otos` sólo enviará una lectura en el caso de que se produzca una nueva.

`Otos` utiliza una librería para acceder a las medidas sensoriales del Pioneer, así como para enviar órdenes de movimiento a los motores. `ARIA` (ActivMedia Robotics Interface Application) es una librería diseñada en un paradigma orientado a objetos para robots móviles de ActivMedia con licencia GPL. Posee dos versiones, una bajo Windows y otra bajo Linux, en nuestro caso, utilizaremos como herramienta la distribución sobre GNU/Linux. `ARIA` incorpora en su plataforma un simulador para el robot Pioneer (SRIsim), como puede verse en la figura 3.10. Una herramienta muy útil ya que permite cargar y crear diferentes mundos sobre los que situar al robot, y simular diferentes casos de prueba de nuestro comportamiento antes de hacerlo sobre el robot real.

`ARIA` ofrece dos tipos de comunicación con un robot, la primera de ellas es a través de puerto serie, de modo que un programa que use `ARIA` y quiera valerse de la accesibilidad que ésta ofrece para comunicarse con el robot Pioneer, debe ejecutarse de manera local en un ordenador a bordo del robot. La otra posibilidad que `ARIA` ofrece, es usar sus clases para comunicarse con el simulador SRIsim a través de conexión TCP/IP, de esta manera, podremos cargar un tipo de robot Pioneer determinado sobre el simulador para realizar lecturas de los sensores y mover al robot en un mundo en concreto.

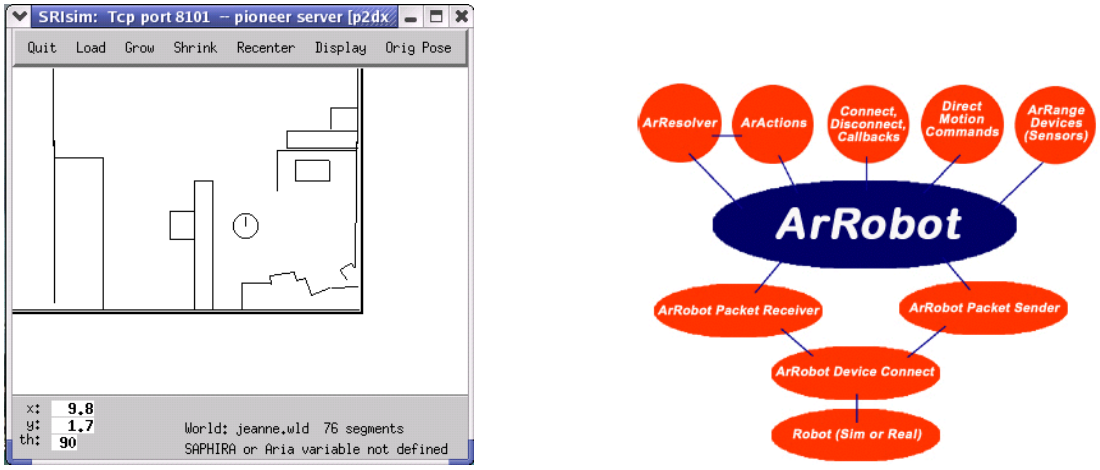

Figura 3.10: Simulador SRIsim y arquitectura software de ARIA.

`ARIA` tiene una limitación, ya que cualquier programa que quiera acceder a las lecturas sensoriales del robot o quiera enviar una órden de movimiento, debe ejecutarse en el PC conectado al robot. De esta forma, `otos` se ejecuta en el PC que viaja a

bordo del robot y usa las clases de ARIA para comunicarse con el Pioneer, y cualquier programa que quiera comunicarse con el robot lo puede hacer a través de `otos` sin necesidad de estar corriendo en el mismo PC.

El otro servidor, `óculo`, se encarga de dar acceso a las imágenes que capta la cámara instalada en el robot. A diferencia de `otos`, `óculo` no envía una nueva imagen a un cliente cada vez que se produce una nueva, sino que en esta ocasión, es el propio cliente quien solicita la imagen cuando le es necesario. De esta forma, contamos con la ventaja de que si un cliente no es capaz de procesar el ritmo de imágenes capturadas que consigue el servidor, éste último no satura al cliente enviándole imágenes según las va capturando, sino que espera a que el cliente le solicite una. En cuanto a la interfaz de mensajes se refiere, y al igual que `otos`, `óculo` posee la suya propia permitiendo a los clientes solicitar imágenes en diferentes formatos, tanto a niveles de gris, como formato RGB.

Como puede verse en la figura 3.9, en el mismo nivel que se encuentra `ARIA` se encuentra `Video4Linux`. `Óculo` accede a las imágenes capturadas por la cámara gracias a la interfaz `Video4Linux`, que incluye el driver que da soporte a la cámara instalada en el robot.

Las comunicaciones entre un cliente JDE y los servidores `otos` y `óculo` se realiza mediante una conexión *tcp*, haciendo uso de los recursos que éstos ponen a su disposición a través de la red, vía sockets. Si como ya hemos visto, esta red es una red inalámbrica, entonces podremos arrancar un cliente en cualquier máquina diferente del PC portátil conectado al robot, y hacer uso de los servicios proporcionados por `otos` y `óculo` de manera remota. En el caso del comportamiento sigue persona, el código se integra bajo uno de estos clientes JDE, suponiendo pues, una herramienta muy útil ya que una gran parte de la base necesaria para generar el comportamiento, dependerá de las medidas ofrecidas por los sensores y de la accesibilidad que nuestros programas puedan tener a los motores.

3.2.3. Esquemas de servicio

Una vez analizada la arquitectura JDE como dos niveles entre los que destacan los clientes JDE (nivel alto), y los servidores `otos` y `óculo` (nivel bajo), vamos a analizar ahora una serie de esquemas, dentro del conjunto de esquemas de un cliente JDE, llamados *esquemas de servicio*. La función de estos esquemas es establecer una comunicación con los servidores, para poner en disposición de los esquemas perceptivos y motores los servicios que `otos` y `óculo` ofrecen. De esta forma, el diseño de nuestro comportamiento no tendrá que ir encaminado a establecer una comunicación directa con estos servidores, sino que la plataforma JDE libera a sus esquemas de percepción y actuación de esa responsabilidad.

Uno de esos esquemas de servicio es `sensationsotos`, que establece una comunicación con el servidor `otos`, recogiendo las medidas sensoriales que éste ofrece, y poniéndolas a disposición del resto de esquemas. Cuando `sensationsotos` recibe esas lecturas, las almacena en variables comunes accesibles a cualquier esquema que quiera hacer uso de ellas. En concreto, guarda en el buffer `laser[i]` las medidas registradas para cada ángulo (de 0 a 180 grados), expresadas en milímetros según la distancia al obstáculo(de 0 a 8000 milímetros). En el caso de los ultrasonidos, la información sensorial se almacena en el buffer `us[i]` según la distribución de la figura 3.11. Las medidas almacenadas en cada posición se refieren a variables de tipo real, que corresponden a la distancia en milímetros obtenida por cada uno de los ultrasonidos, en concreto, estas variables oscilan entre 0 y 3000 milímetros, ya que los ultrasonidos del pioneer pueden llegar a medir hasta tres metros de distancia.

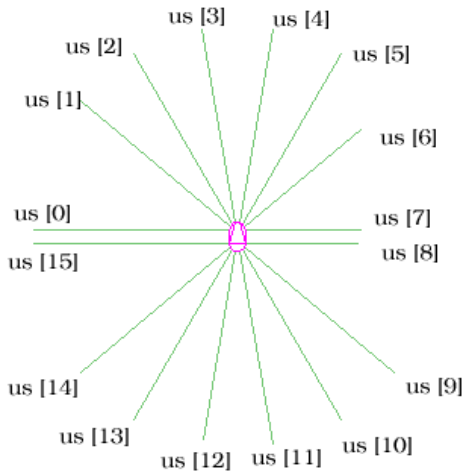

Figura 3.11: Distribución de los ultrasonidos en el array us[]

Otro esquema de servicio es `sensationsoculo` y su labor consistirá en solicitar la información sensorial que recoge `óculo` para ponerla a disposición del resto de esquemas. Si `sensationsotos` almacenaba los valores de los sensores recogidos por `otos` en ciertas variables compartidas, en el caso de `sensationsoculo` sucede lo mismo. En esta ocasión, `sensationsoculo` almacena la imagen que viene de `óculo` en un buffer `colorbuf[]` si la imagen es en color, o `mmbuf[]` si la imagen está en niveles de gris. Para el caso de la imagen en color, dicho búffer tendrá el tamaño de 320x240x3 posiciones. Este tamaño corresponde a una imagen RGB de 320 pixeles de ancho, 240 pixeles de largo, con 3 bytes por cada pixel, uno para la componente roja R, otro para la componente verde G y otro para la componente azul B. También hay que tener en cuenta que la imagen se almacena en el array partiendo de la base de que el pixel de coordenadas (0,0) se encuentra en la parte superior izquierda, y que la lectura se realizará por filas.

La plataforma JDE aún ofrece dos esquemas más de servicio, uno de ellos es el esquema `motors`. La tarea de este esquema es la de establecer una comunicación con el servidor `otos` para enviárle las órdenes de movimiento que los esquemas motores quieren realizar sobre el robot. Para ello, cuando un esquema motor quiere enviar una órden de movimiento a los motores del robot, sólo tendrá que modificar dos variables `v` y `w` que corresponden a la velocidad lineal y angular respectivamente. El esquema `motors` enviará esos parámetros al servidor `otos` para que los ejecute sobre el robot a través de la interfaz de mensajes con *OTOS_drive_speed(velocidad,aceleración)*, variando, en este caso, la velocidad de tracción dependiendo de los parámetros (velocidad, aceleración), sin enviar mensaje de confirmación de orden ejecutada.

El otro esquema de servicio muestra un interfaz gráfico que mejora la interacción con los usuarios. Este interfaz es gobernado por otro esquema llamado `guixforms` y nos muestra imágenes, recogidas por la cámara, en servidores X con diferentes profundidades: 8 bits, 16 bits y 24 bits, así como las diferentes sensaciones recogidas por el robot. También permite activar o desactivar un esquema creado según nos convenga, así como parar o poner en marcha los motores cuando se desee, lo que la convierte, dentro de lo que es la plataforma JDE, en una herramienta sobre la que poder depurar los esquemas de nuestro comportamiento. En la figura 3.12 se muestra un ejemplo de esta interfaz, dibujando en la ventana de la parte superior, lo recogido por el láser y mostrando el alcance de los sonares, así como las imágenes de la cámara en la ventana inferior. En los lados de la interfaz, los esquemas que podemos activar y desactivar a nuestro antojo. Para este ejemplo, activamos los esquemas que muestran el láser, los sonares y la imagen en color.

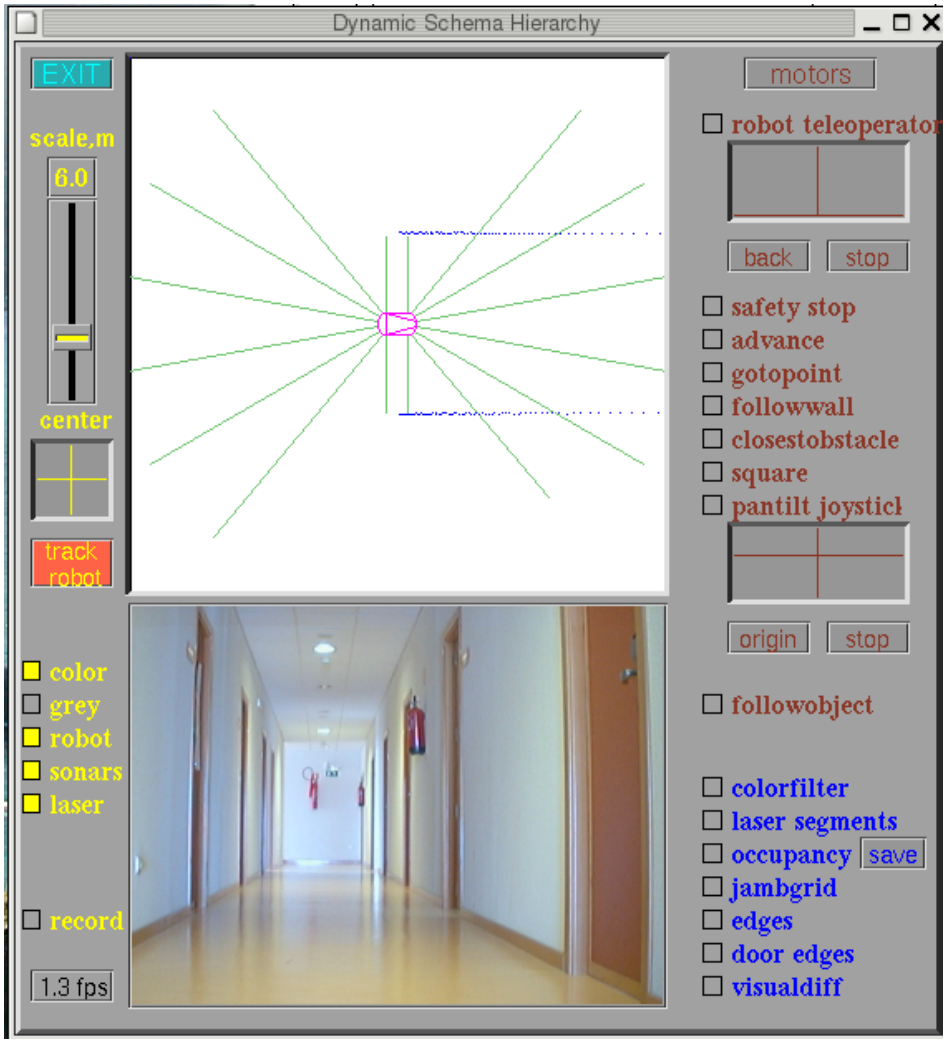

Figura 3.12: Visualización del esquema Guixforms

Capítulo 4

Descripción Informatica

Hasta ahora hemos visto los objetivos que, según nuestro enfoque, ha de cumplir un comportamiento sigue persona con visión. Hemos analizado en menor o mayor medida cada una de las herramientas de las que disponemos para llevarlo a cabo, tanto en la parte hardware como en la parte software. Sabemos *qué* hay que hacer, y las herramientas que tenemos para realizarlo, pero aún no hemos hablado de *cómo* se genera el comportamiento sigue persona. En este capítulo se describimos precisamente ésto último, dando primero una descripción global del problema, y la forma de resolverlo con dos esquemas de JDE. También se detallarán esos esquemas, uno de ellos encargado de la parte de visión llamado esquema perceptivo `colorfilter`, y otro encargado de la parte de actuación denominado `followobject`. Para terminar este capítulo, se muestran algunos resultados experimentales llevados a cabo en diferentes escenarios.

4.1. Diseño global

En este apartado damos una descripción, a grandes rasgos, de cómo se ha diseñado el comportamiento sigue persona, teniendo en cuenta que el objetivo del comportamiento es que un robot siga a una persona vestida con ropa llamativa sin chocar contra obstáculos.

Ya presentamos en el capítulo dedicado a las herramientas a la plataforma JDE, donde se integra el código que genera este comportamiento. Hablábamos de un cliente de JDE como estructura un conjunto de esquemas concurrentes, cada una asociada a una tarea específica y ejecutándose de manera iterativa. Separábamos estos hilos de ejecución en esquemas perceptivos y esquemas motores, de tal forma, que su naturaleza iterativa nos permite enviar varias órdenes de movimiento al robot por segundo.

Para el comportamiento sigue persona con visión se han diseñado dos esquemas que se integran al conjunto de los ya existentes dentro de la arquitectura de JDE. Uno de ellos perceptivo como es `colorfilter`, y el otro motriz, como es `followobject`, cuya ejecución concurrente en el robot genera este comportamiento.

El esquema `colorfilter` es el encargado de analizar la imagen con el fin de

comprobar si hay o no objetivo que seguir, y en caso de que lo haya, calcular su posición en la imagen. Este esquema cuenta con la ayuda de que la persona va vestida con ropa llamativa, en concreto, con una camiseta de color verde. Para llevar a cabo esta función, `colorfilter` realiza un filtrado de color de la imagen, y así separa aquello que le interesa seguir del resto de objetos que capta la cámara. Una vez distingue lo que es objetivo a seguir de lo que no, calcula una coordenada x,y que corresponderá a la posición que ocupa la persona en la imagen.

Por su parte, el esquema `followobject` es el encargado de enviar las órdenes de movimiento al robot, tomando como referencia lo analizado por el esquema perceptivo. Es decir, si `colorfilter` localiza a la persona en la parte derecha de la imagen, `followobject` deberá tomar esta información para modificar los parámetros de velocidad lineal y angular del robot, con la intención de dirigirse hacia ese lado (figura 4.1).

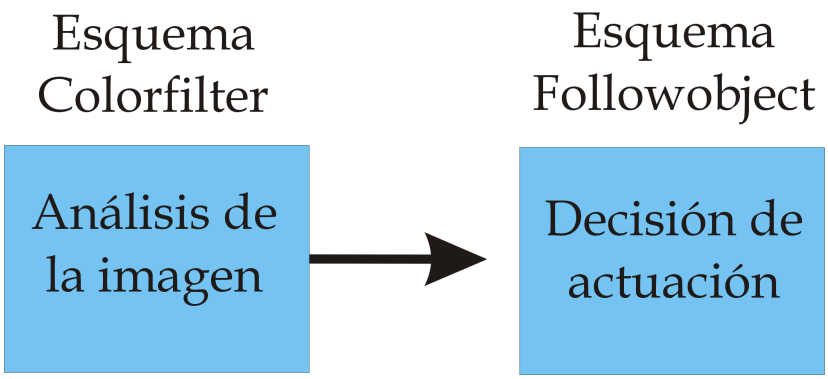

Figura 4.1: Esquemas implementados para el comportamiento

La tarea del esquema motriz no sólo irá enfocada a variar la velocidad del Pioneer, sino que antes de hacerlo deberá comprobar si puede moverse hacia esa dirección, es decir, comprobar que no hay obstáculos con los que pueda chocar si sigue moviéndose en ese sentido y a esa velocidad. Además, tiene en cuenta qué hacer si pierde de vista a la persona, o lo que es lo mismo, que hacer si el esquema perceptivo no encuentra la camiseta verde en la imagen. Para ello, el esquema `followobject` lleva a cabo un *control basado en casos*, de manera que para cada caso aplica una regla de control. Es decir, si hay obstáculos cerca modera su velocidad, si no hay obstáculos aumenta su velocidad, si la persona está muy cerca se para...

La comunicación entre los diferentes esquemas se realiza de modo asíncrono a través de variables compartidas. De modo que el esquema `colorfilter` recoge la información necesaria a través de un búffer en el que `sensationsoculo` ha almacenado la imagen en color procedente del servidor `óculo`, efectúa el correspondiente análisis para calcular la posición de la persona y guarda estos datos en variables compartidas.

Por su parte, el esquema motriz `followobject` hace uso de estas variables actualizadas por el esquema perceptivo para tomar una decisión de control, analizando antes las medidas procedentes de los sensores del robot para saber si le es necesario corregir la trayectoria. Similar al caso del esquema perceptivo, el esquema motriz recoge la información procedente de los sensores a través de `sensationsotos`, el cual cada vez que le llega una nueva lectura del servidor `otos`, actualiza una serie de variables comunes a todos los esquemas de JDE. Una vez determinada la trayectoria a seguir, el esquema motriz varia las variables correspondientes a la velocidad del robot y el esquema de servicio `motors` de JDE se encarga de enviárselas al servidor otos para que las ejecute sobre el robot.

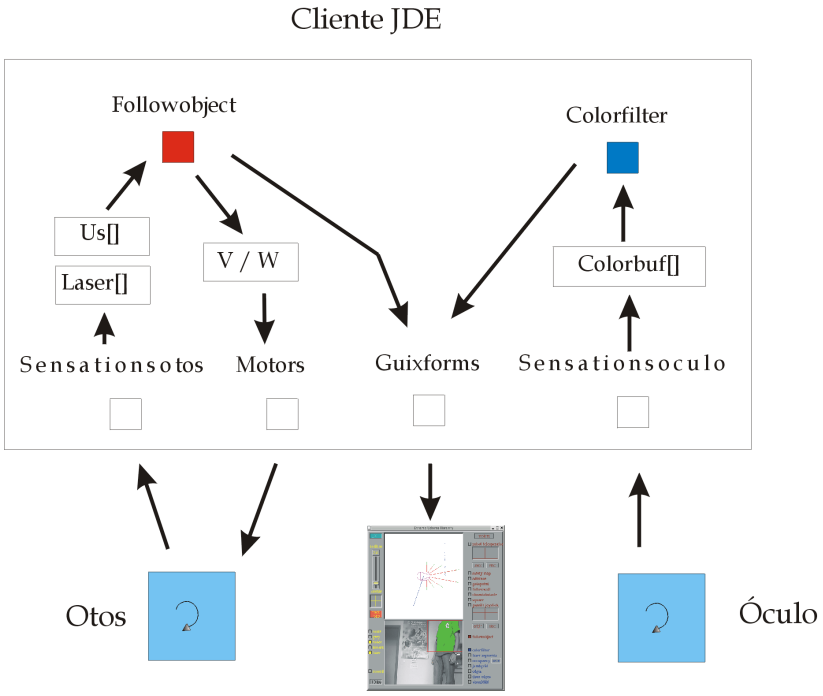

Figura 4.2: Comunicacion con esquemas de servicio

4.2. Esquema Perceptivo

En esta sección se habla en detalle del esquema `colorfilter`, analizando cada una de las fases que comprenden la tarea que desempeña.

Esta tarea corresponde a un análisis de la imagen, de la que el esquema perceptivo trata de sacar dos informaciones básicas: primero, la existencia o no de persona en la imagen, y segundo, en el caso de que exista persona en la imagen, conocer su posición. Estos datos son almacenados de forma que el resto de esquemas pueda hacer uso de ellos cuando lo requiera. En concreto, estos datos serán usados posteriormente por el esquema motriz como entrada de su iteración para decidir el movimiento del robot.

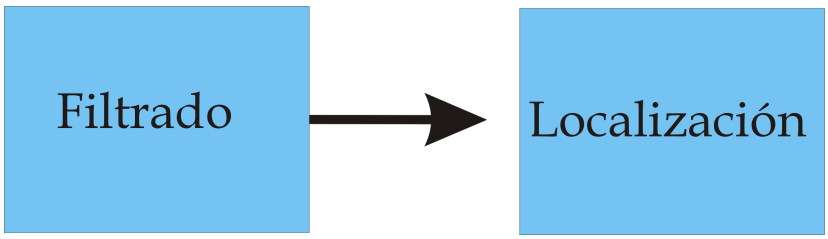

Figura 4.3: Etapas del esquema perceptivo

El esquema `colorfilter` recibe como entrada una imagen en formato RGB, y tras aplicar un filtro de color devuelve otra imagen con una serie de pixeles marcados que han pasado el filtro. Poniéndo a disposición del resto de esquemas la imagen filtrada, el centro de masas de los pixeles marcados y el número de pixeles que pasan el filtro.

El motivo de usar una camiseta de color llamativo, se debe a que el planteamiento de este proyecto trata de centrarse en el seguimiento y no tanto en la percepción. También veremos como este filtrado no siempre se aplica sobre todos los píxeles de la imagen.

4.2.1. Filtrado por color

La imagen que el esquema perceptivo analiza está almacenada en una variable común llamada `colorbuf[]`. Esta variable es actualizada por el esquema de servicio `sensationsoculo`, que recoge esta información a través de una comunicación directa con el servidor óculo. De esta forma, el diseño de nuestro esquema perceptivo, comienza con la lectura de esa variable, que contendrá la imagen en color en formato RGB, y con una resolución de 320x240 pixeles.

El esquema `colorfilter` tiene a su disposición la imagen en formato RGB procedente de la cámara, la cual analiza con el propósito de mejorar la funcionalidad del resto de etapas que intervienen en el seguimiento. El filtro recibe como señal de entrada la imagen capturada y obtiene una imagen con la persona identificada (4.5).

El espacio sobre el que se realiza este filtro de color es el espacio HSI. Este espacio posee tres componentes: La componente H, que es el Matiz del pixel, o lo que es lo mismo, al color en sí; la componente S, que es la Saturación, es decir, identifica la claridad del color; y la componente I, que es la Intensidad del color o luminosidad (figura 4.4). Al aplicar un filtro en este espacio obtenemos una mayor robustez ante cambios de luminosidad, ya que podemos ignorar la componente I. De esta manera, lo primero que efectúa el esquema `colorfilter` es una conversión de espacio RGB a espacio HSI siguiendo las ecuaciones:

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

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

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

Donde R corresponde a la componente roja, G a la componente verde, y B a la componente azul de la imagen origen en formato RGB. Así como la función *min(R,G,B)* devuelve el mínimo valor de las tres componentes.

Una vez tenemos la imagen en el espacio HSI, definimos un rango de valores correpondientes a cada una de las componentes del HSI, exceptuando como ya hemos dicho a la componente I para ignorar los cambios en la luminosidad. Este rango de valores vendrá dado por un *Hmaximo* y un *Hminimo*, así como un *Smaximo* y un *Sminimo* definidos para el color llamativo de la camiseta de la persona. Para los experimentos hemos elegido un color verde cuyos valores se muestran en la tabla 4.1. Para realizar el filtrado, se analiza pixel a pixel la imagen, comprobando que el valor de matiz y saturación de cada pixel entra dentro del rango de valores H y S establecidos. Si cumple estas condiciones, se satura la componente verde de ese pixel, dejando la componente roja y azul a valor cero. Si por el contrario, el pixel no cumple estos requisitos, se dejarán sus valores RGB originales.

Valores H y S para el filtro

	Rango de valores			
	Hmaximo	Hminimo	Smaximo	Sminimo
Camiseta	181.60	170.50	0.90	0.39

Cuadro 4.1: Rango de valores HS para la camiseta verde

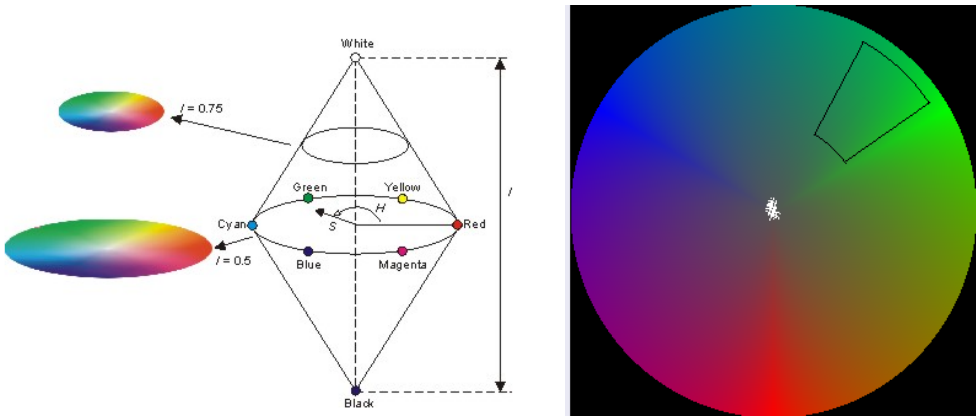

Figura 4.4: Representación del modelo de color HSI (izquierda) y rango de valores para la camiseta (derecha)

Al final del proceso, obtenemos una imagen de salida almacenada en `colorfiltermmbuf[]`, de dimensión y formato igual al de la imagen de entrada, pero con una serie de pixeles *"marcados"* de color verde (componente R=0, componente

G=255, componente B=0) que identifican a la persona, y el resto en blanco y negro, como vemos en la figura 4.5.

Figura 4.5: Ejemplo de la imagen tras aplicar el filtro HSI

El esquema `colorfilter` además de identificar a la persona, calcula su posición en la imagen. Para ello calcula el centro de masas de los pixeles de la camiseta, que vendrá definido por un pixel de coordenadas x e y en la imagen, obtenidas de la siguiente manera:

$$\widehat{x} = \frac{\sum_{i=1}^{P} X_i}{P}$$

$$\widehat{y} = \frac{\sum_{i=1}^{P} Y_i}{P}$$

siendo P el total de pixeles marcados.

Estas variables son accesibles para el resto de esquemas, y en nuestro caso, como veremos más adelante, utilizadas por el esquema motriz para llevar un control de la velocidad del robot.

Durante este proceso de análisis, surgieron varias dificultades al realizar los experimentos sobre la imagen que pasamos a describir a continuación.

El primer problema que se nos presenta a la hora de efectuar la localización de la persona es la robustez frente a cambios de iluminación, ya que el robot en su proceso de seguimiento, atraviesa zonas de iluminación muy variables, y el filtro ha de ser lo suficientemente robusto en estos casos. En un principio, y dado que la imagen almacenada por sensationsoculo en `colorbuf[]` está en formato RGB, se implementó un filtro en este espacio. Con los experimentos en la cámara real, se comprobó que el filtro sobre RGB no era fiable, era muy sensible con respecto a los cambios de iluminación, por lo que se realizaron las pruebas bajo el espacio HSI, tomando como referencia el proyecto *Filtro de color configurable* [Matute03].

Una de las características técnicas de la cámara usada en este comportamiento también supuso un problema. Esta cámara tiene un ajuste automático de color (autoiris), siendo diferente el valor RGB de los pixeles de un objeto que aparece de repente en la imagen, al valor RGB de los pixeles pasado un tiempo que el objeto es captado por la cámara. Esta variación también afecta a los valores HS del pixel, aún así el filtro de color HSI está implementado de manera que soporta este autoiris de la cámara, siendo lo suficientemente robusto ante estos cambios en el color.

Otro problema se presentó cuando, sin estar la persona en la imagen, un cierto número de pixeles desagrupados superaban el filtro, con lo que se realizaba un cálculo no deseado del centro de masas. Es conveniente que este centro se calcule sólo cuando la persona esté en la imagen. La manera que tiene el esquema `colorfilter` de diferenciar entre cuando hay persona en la imagen y cuando no es a través de un *número mínimo* de pixeles. De tal forma, que el cálculo del centro de masas se realiza sí y sólo sí el número pixeles que han pasado el filtro es mayor a ese *número mínimo*. Tras varios ajustes experimentales, esta umbral de situó en 1000 pixeles.

4.2.2. Ventana de atención

No obstante, este número mínimo de pixeles puede superarse, como es el caso en que además de la persona, aparecen más objetos del mismo color de la camiseta verde, calculando un centro de masas que no corresponde a la posición real de la persona en la imagen. Una posible solución a este problema es separar en "zonas" o "ventanas" los diferentes pixeles marcados mediante una segmentación, como en el *Comportamiento sigue pelota con visión local* [SanMartín02], en cuyo caso se aplica una segmentación sobre la imagen para separar los objetos en diferentes zonas. Ésto requiere analizar todos los pixeles de la imagen en cada iteración, lo que supone que el esquema perceptivo, en nuestro caso, sólo es capaz de procesar de 2 a 3 imágenes por segundo. Con esta frecuencia de análisis, el comportamiento no consigue cumplir uno de sus requisitos (capítulo 2), la vivacidad.

El otro problema se presentó en el apartado anterior, y se plantea a la hora de calcular el centro de masas, ya que los pixeles pueden aparecer desagrupados debido, por ejemplo, a que en la imagen existan varios objetos del mismo color que la camiseta, pero en diferentes posiciones, lo que nos daría un centro de masas erróneo.

Para el comportamiento sigue persona se creó una *ventana de filtrado* sobre cuyos pixeles se aplica el filtro de la tabla 4.1. De modo que los pasos a seguir por el esquema `colorfilter` son:

1 Si la persona no está dentro del campo visual, se aplica el filtro sobre todos los pixeles de la imagen hasta que se localize a la persona.

2 Una vez se la identifica y se calcula su posición a través del centro de masas, en la

siguiente iteración, se toma como referencia este pixel para hallar las coordenadas que formarán la ventana sobre cuyos pixeles se aplicará a partir de ahora el filtro. En las sucesivas iteraciones, se calculan nuevas ventanas que tendrán como punto central el centro de masas hallado en la iteración anterior.

3 Si la persona vuelve a desaparecer de la imagen, el esquema `colorfilter` aplica el filtro sobre todos los pixeles de la imagen volviendo al paso 1.

Con esto se consigue acelerar el proceso de filtrado, y por tanto, que el esquema perceptivo sea capaz de analizar muchas más imágenes por segundo. En concreto, en los experimentos realizados sin la aplicación de esta ventana de atención, obteníamos una velocidad de análisis de 2 a 3 imágenes por segundo. Al implementar este tipo de ventana, la velocidad pasó a ser de 8 a 9 imágenes analizadas por segundo, con lo que se consigue que el comportamiento cumpla uno de sus objetivos, la vivacidad.

El tamaño de la ventana sobre la que se realizaron los experimentos fue de 160x120, definido en un par de constantes en el código como *Ventana alto* y *Ventana ancho*, que se mantiene fijo durante toda la ejecución. En la figura 4.6 observamos la imagen original, y la ventana que se crea tras aplicar el filtro HSI.

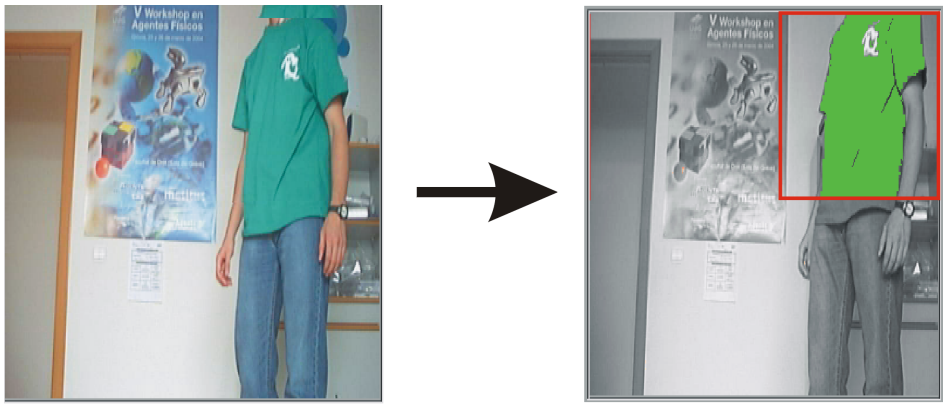

Figura 4.6: Creacion de la ventana de atención sobre la imagen

También se produce una diferencia entre los experimientos con y sin ventana en lo que al cálculo del centro de masas se refiere. Sin el uso de esta ventana, el cálculo del centro de masas se puede efectuar sobre una imagen en la que, además de los pixeles de la camiseta, existen otros pixeles sueltos que pasan el filtro, y que dan lugar a un centro de masas que no se corresponde con la posición real de la persona. Con la incorporación de la ventana de atención, no sólo se reduce el error de calcular un centro de masas que no se ajusta a la posición de la persona en la imagen, si no que además este centro de masas define de manera más exacta esta posición.

La plataforma JDE permite acotar la frecuencia máxima de un esquema. De esta manera, se puede regular el número de iteraciones que ejecuta en un segundo. Cuantas

más iteraciones por segundo ejecute el esquema `colorfilter`, más imágenes será capaz de analizar, mejorando la vivacidad del comportamiento.

Con el fin de poder depurar mejor nuestro código, y ver si el filtro se ejecuta de manera correcta, el esquema de servicio `guixforms` del que ya hablamos en el capítulo 2, permite visualizar la imagen resultante del filtro con la ventana de atención sobre el que ha sido aplicado. En la figura 4.16 podemos ver el resultado, bajo una interfaz JDE, después de ejecutarse el esquema `colorfilter` sobre la imagen.

4.3. Esquema Motriz

Gracias al esquema perceptivo `colorfilter` hemos detectado y localizado a la persona en la imagen. El esquema motriz se encarga de, en función de esa información, mover al robot llevando a cabo un control en velocidad con el fin de seguir a la persona. Además de la información exraída por el esquema perceptivo, el esquema motriz recibe como señal de entrada los datos sensoriales procedentes del sensor láser y los sónares del robot. Con ello, genera una señal de control basada en casos que persigue:

1 Perseguir a la persona.

2 No chocar contra obstáculos.

El esquema motor define dos zonas, una de seguridad y otra de precaución, y dependiendo de si estas zonas detectan obstáculos, generará una señal de control u otra. También la decisión de control que toma el esquema `followobject` varía en función de si la persona está o no en la imagen. En la tabla 4.2 se presentan algunos de los casos que pueden darse a la hora de crear un comportamiento sigue persona. Estos casos serán analizados posteriormente en detalle.

4.3.1. Seguimiento del objetivo

`Followobject` lee las variables compartidas `pixelcentralx` y `pixelcentraly` que el esquema `colorfilter` se encarga de actualizar, y que corresponden a las coordenadas de la posición aproximada en la que se encuentra la persona en la imagen, estableciéndose así una comunicación asíncrona entre ambos hilos de ejecución.

Lo primero que efectúa el esquema motor tras leer las variables correspondientes al pixel central es una asociación, o lo que es lo mismo, aplica un *modelo geométrico* que realiza la conversión de esas coordenadas a un determinado ángulo. Este ángulo se refiere a la dirección en la que se encuentra el objetivo a seguir, y se realiza para posteriormente alinear al robot con la persona. La cámara está montada apuntando al frente del robot y tiene un cono de visión de 60 grados en horizontal. Sabiendo la posición del centro de la camiseta se puede estimar su orientación con respecto al robot,

Caso	*Acción*	*Control*
Zona seguridad libre Zona precaución libre Persona presente	Seguimiento a máxima velocidad lineal según dirección persona	v y w dependen de la dirección de la persona.
Zona seguridad libre Zona precaución ocupada Persona presente	Seguimiento con precaución según dirección persona	v se reduce, w según dirección persona.
Zona seguridad ocupada Zona precaución ocupada Persona presente	Cálculo de dirección alternativa que evite obstáculo Sorteo de obstáculos.	v y w dependen de la nueva dirección.
Perdida de la persona	¿En que dirección la vimos por última vez? Búsqueda.	v y w dependen de la última dirección.
Caso especial de búsqueda. Persona desaparece por zona superior centrada de la imagen	la persona se encuentra muy cerca	v y w igual a cero. El robot se para.

Cuadro 4.2: Casos del comportamiento sigue persona

aplicando este modelo geométrico. El resultado será un ángulo comprendido entre 60 y 120 grados, en el sistema de referencia del robot, que se almacena en la variable *gradocentroimagen* como vemos en la figura 4.7.

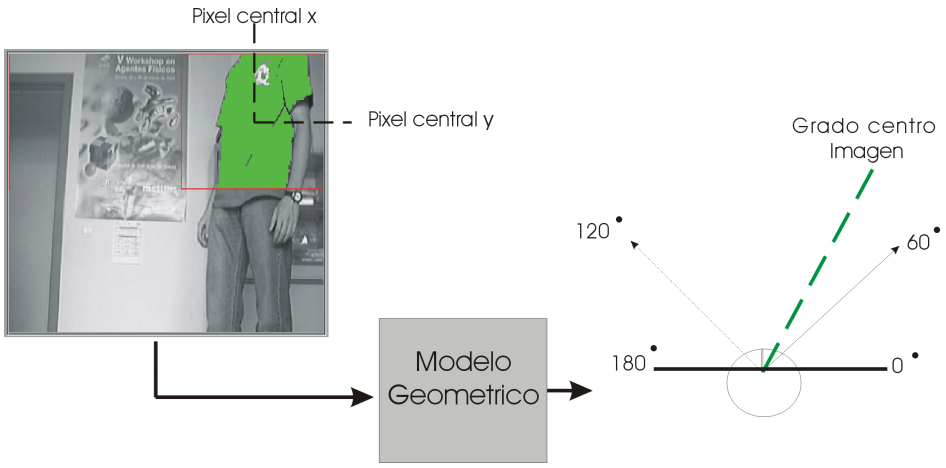

Figura 4.7: Aplicacion del modelo geometrico sobre la imagen

De esta manera, ya no tenemos solamente localizada a la persona y conocemos su posición en la imagen, sino que también sabemos la dirección que ocupa con respecto al robot, dado el ángulo que forma con él. La tarea de este esquema es la de mover al robot en una determinada dirección donde se encuentre la persona. Por tanto, se puede tomar como referencia el ángulo devuelto por el modelo geométrico para seguir esa dirección, ya que será la dirección de avance deseada en el caso que no haya obstáculos, o una próxima a ella en el caso de que los haya.

4.3.2. Zona de seguridad

Después de aplicar el modelo geométrico, tenemos una dirección a seguir. A priori se podría pensar que lo siguiente a realizar por el esquema motriz es, como su propio nombre indica, enviar órdenes de movimiento a los motores del robot, pero antes de eso, se ha de comprobar que en la dirección propuesta no hay obstáculos.

Para ello el esquema motriz lee de la variable `us[i]` las medidas recogidas por los ultrasonidos, y de la variable `laser[i]` las medidas recogidas por el sensor láser del robot en un determinado instante. Estas variables, como vimos en el capitulo 3, son actualizadas por el esquema de servicio `sensationsotos`.

Con lo que llegado a este punto, el siguiente paso consiste en analizar el entorno a través de las medidas sensoriales de los sónares frontales y del sensor láser, para comprobar que podemos efectuar el seguimiento sin chocar con ningún obstáculo. Para ello se definen dos zonas, las cuales indicarán al robot si, puede ir en la dirección de la persona acelerando paulatinamente hasta alcanzar la máxima velocidad, o por el contrario ha de modificar esa dirección para evitar algún obstáculo.

En concreto se crean dos zonas, una llamada *zona de seguridad* y otra llamada *zona de precaución* . La amplitud de cada zona de seguridad y precaución viene definida por dos datos de entrada como son la *distancia frontal* y *distancia lateral*. Estas variables de *distancia lateral* y *distancia frontal* son diferentes dependiendo de la zona que queramos tratar. En el caso de la zona de seguridad, la *distancia frontal* será mayor que en el caso de la zona de precaución, teniéndo ésta última mayor *distancia lateral* que la zona de seguridad (4.8).

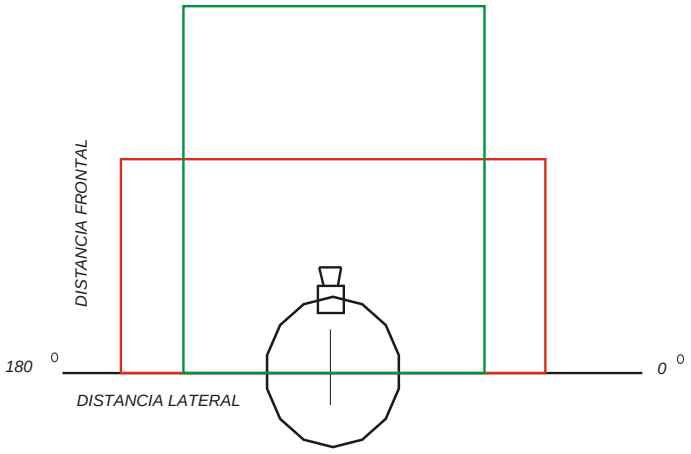

Figura 4.8: Zonas de seguridad(verde) y precaución(rojo)

Estas dos zonas vienen definidas por una serie de medidas establecidas llamadas *distanciaslimite*, calculadas a partir de la *distancia lateral* y la *distancia frontal* como vemos en la figura 4.9.

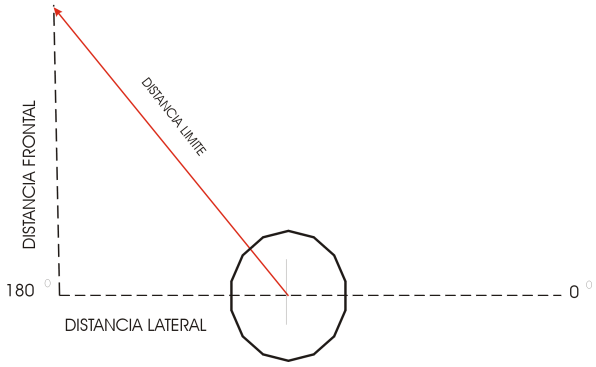

Figura 4.9: Calculo de la distancia límite

Formándose así cada una de las zonas (figura 4.11,4.10) definidas por rectángulos. Las distancias límite son almacenadas por el esquema motriz en dos buffers llamados *sonar rectangulo seguridad* y *sonar rectangulo precaución*. En el caso de las zonas definidas para los ultrasonidos estos buffers será de ocho posiciones, y cada posición contendrá la distancia límite de 0 a 3000 asociada a un determinado sonar delantero. Para el caso del sensor láser, estas distancias límite se almacenan en *laser rectangulo seguridad* y *laser rectangulo precaución*, siendo estas variables buffers de 180 posiciones, cada una asociada a un determinado ángulo y que contienen las distancias límite para los láser.

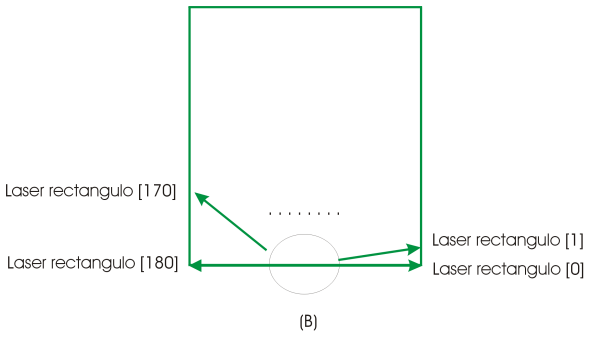

Figura 4.10: Distancias límite para el sensor láser

Una vez definidas las variables *distancia frontal* y *distancia lateral*, la dimensión de ambos rectángulos es fija durante toda la ejecución, de modo que el cálculo de estas distancias límite se realiza en la primera iteración, ahorrando así tiempo de cómputo en el resto de iteraciones.

Estas zonas se usan a modo de *líneas frontera*. De modo que si alguna de las medidas reales recogidas por el sensor, es menor que su correspondiente distancia límite, significa que algún obstáculo ha traspasado esa línea frontera, considerándose a esa zona como *zona no segura*.

1 .Si us[i] es $\geq$ que sonar rectangulo seguridad[i] entonces zona segura

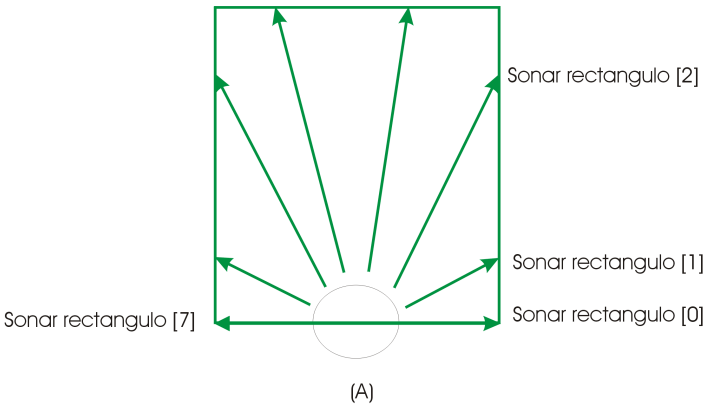

Figura 4.11: Distancias límite para el sensor sonar

2 .e.o.c. zona no segura

de igual forma se aplicaría al sensor láser.

4.3.3. Control basado en casos

A continuación se analiza la forma en que el esquema motriz toma una decisión de control para llevarlo a cabo sobre el robot. Esta decisión de por dónde y hacia dónde se va a mover el robot depende del caso en el que nos encontremos. Estos casos vienen reflejados en la tabla 4.2 y algunos de ellos se resuelven de manera sencilla, pero en otros su solución requiere una serie de cálculos.

1. Empezaremos por centrarnos en los casos que se resuelven de manera sencilla. El primero de ellos se presenta cuando no se detectan obstáculos ni en la zona de seguridad ni en la zona de precaución (*caso sin obstáculos*). En este caso el robot acelera hasta alcanzar la máxima velocidad lineal establecida en dirección a la persona, que vendrá definida por el ángulo almacenado en la variable *gradocentroimagen*.

2. Un segundo caso se produce cuando algún obstáculo se encuentra en la zona de precaución, pero sin embargo no invade el rectángulo de seguridad (*caso obstáculos alejados*). Ésto le indica al esquema `followobject` que aunque no hay obstáculos que le impidan seguir hacia el objetivo, sí se encuentran lo suficientemente cerca como para que la velocidad de seguimiento sea moderada. Con lo cual el robot sigue guiándose por la posición de la persona en la imagen, pero sin alcanzar grandes velocidades.

3. Si las dos zonas han detectado algún obstáculo ya no sirve moderar la velocidad, sino que ahora es necesario calcular una dirección alternativa que evite el obstáculo (*caso obstáculos en las proximidades*). El cálculo de esta nueva dirección (evitación del obstáculo) que detallamos a continuación se basa en las medidas de

los sónares, ya que las medidas ofrecidas por el sensor láser no pudieron utilizarse finalmente por problemas de hardware de este dispositivo.

El criterio que sigue es el de *calcular una trayectoria lo más alejada del obstáculo y que a su vez no pierda de vista a la persona.* Este cálculo se realiza a través de dos datos de entrada, el primero de ellos pertenece a las medidas sónares almacenadas en `us[]`, y el segundo corresponde al ángulo objetivo devuelto por el modelo geométrico (*gradocentroimagen*), y de cuya dirección debemos tratar de alejarnos lo menos posible.

La idea para calcular una dirección que evitara el obstáculo, y que a la vez no se alejara mucho de la persona la obtuvimos de los `Campos Virtuales de Fuerza (VFF)`. [Borenstenin89]. Este método suma vectorialmente una dirección objetivo (*fuerza atractiva*) a la que el robot quiere ir, y una dirección que evita el obstáculo (*fuerza repulsiva*). El resultado es una dirección (*fuerza resultante*) que evita el obstáculo y no se desvía demasiado de la dirección objetivo.

En esta misma línea, el algoritmo desarrollado en este proyecto para el esquema `followobject`, lo primero que hace es hallar la mínima distancia registrada por los sónares frontales con el propósito de saber porque lado está más cerca el obstáculo. Una vez conoce la posición del sonar que tiene esa mínima distancia, también conoce el ángulo que forma con el robot, y dependiendo del valor de esta distancia se calcula la dirección que debería tomar el robot para evitar chocar. Esta dirección vendrá dada por un ángulo comprendido entre 0 y 2π calculado de la siguiente manera:

$$\Theta_{repulsivo} = \Theta_{minmedida} \pm \left(\pi - \frac{distanciaminima}{distanciamaxima} * \pi \right)$$

teniendo en cuenta la discontinuidad a 2π.

Donde $\Theta_{minmedida}$ corresponde al ángulo que forman el robot y el sonar que ha registrado la mínima medida, *distanciaminima* al valor de esa mínima medida en milímetros, y *distanciamaxima* a la máxima distancia que pueden llegar a medir los ultrasonidos, es decir 3000 milímetros.

De esta forma, si la medida mínima resgistrada por el ultrasonido estuviera próxima a cero, la amplitud del ángulo formado entre $\Theta_{minmedida}$ y $\Theta_{repulsivo}$ estaría muy próxima a 180 grados, amplitud que se íria recortando a medida que la mínima distancia recogida por el sonar va siendo mayor, para no desviarnos de la dirección en la que se encuentra la persona.

Con esto conseguimos generar una dirección, que en el caso de que el obstáculo este demasiado cerca del robot, será totalmente contraria a la dirección donde

se ha registrado esa mínima distancia. En la figura 4.12, podemos observar la orientación de la *fuerza repulsiva*, que se aleja de la dirección en la que se ha registrado la mínima medida sonar, que en este caso marcaría que el obstáculo está muy cerca del robot.

Una vez el esquema ha generado el ángulo de la *fuerza repulsiva*, lo siguiente que se ha de tener en cuenta es el no perder de vista a la persona, a no ser que sea necesario. Esto se realiza sumando vectorialmente el ángulo de la *fuerza repulsiva* y el ángulo de la *fuerza atractiva*, de modo que se calcula un ángulo que marcará una nueva dirección alejada del obstáculo, y a su vez, no se aleja demasiado de la dirección en la que se encuentra la persona (figura 4.12).

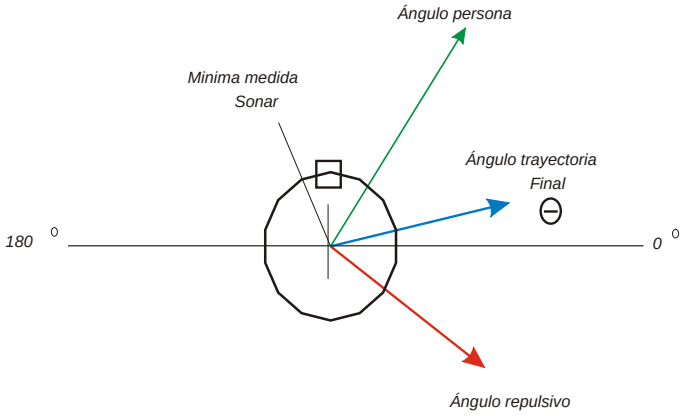

Figura 4.12: Calculo de la trayectoria que evita el obstáculo

4. Cuando el robot pierde de vista a la persona, ya bien sea por la derecha, por la izquierda o por el centro, se ha programado una búsqueda para que el robot trate de encontrarla (*Caso de búsqueda*).

   En cuanto se pierde de vista a la persona, el esquema motriz analiza la posición en la que se encontraba en el momento anterior, es decir, se basa en el último centro de masas hallado por el esquema perceptivo. Si esta posición estaba orientada hacia la izquierda de la imagen, el esquema `followobject` moverá al robot en esa dirección, y análogamente si la posición se encontraba orientada hacia la derecha. Esta dirección depende de en que franja de la figura 4.13 se sitúe el último centro de masas calculado.

   Si está en la franja de *búsqueda izquierda*, la dirección que genera el esquema motriz coincide con un ángulo de 135 grados, con intención de dirigir al robot hacia la izquierda para que realice la búsqueda. Si está en la franja de *búsqueda derecha*, la dirección coincide con el ángulo 45 grados.

Para la franja central, suponemos que es muy probable que la haya perdido de vista por una fallo del filtro HSI, debido a algún cambio brusco de luminosidad, con lo que la dirección elegida irá en línea recta hasta que las condiciones de luminosidad sean favorables. En cualquiera de los tres casos, el esquema motriz no deja de analizar las condiciones del entorno con el fin de evitar cualquier obstáculo durante su labor de búsqueda. Si en cualquier instante, la persona volviese a ser visualizada en la imagen, el robot comenzaría de nuevo su proceso de seguimiento natural. Si pasado un *tiempo de búsqueda* el robot no encuentra a la persona, se para hasta que vuelva a aparecer en la imagen.

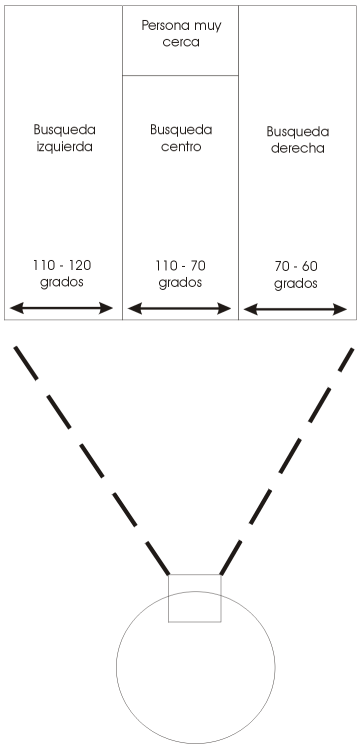

Figura 4.13: Definición de zonas de búsqueda en la imagen

5. El robot tiene que detenerse cuando realizando el seguimiento, se acerca a la persona (*caso de proximidad*).

   La solución que se plantea forma parte de un caso especial de búsqueda, ya que aprovechamos cuando el robot pierde de vista a la persona por una zona central de la imagen, para determinar que está muy cerca de ella. Definimos otra franja en la parte superior centrada de la imagen(4.13). Dado que el robot siempre tiende a mantenerse alineado con la persona, y conociéndo la posición y orientación de la cámara en el robot, consideramos que tiene muy cerca a la persona, cuando la posición que ocupaba en la imagen en el instante anterior se encontraba dentro de esa franja. Para ello contamos con que la persona es suficientemente alta como para que el robot la pierda de vista por esa franja cuando está a cierta distancia

de ella (4.14).

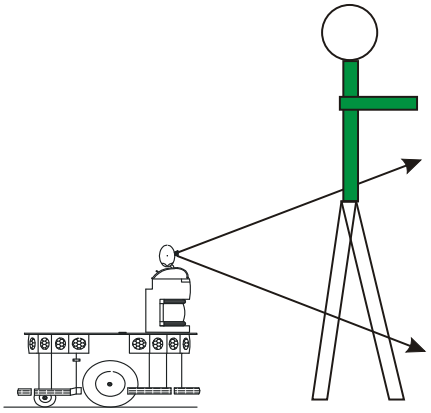

Figura 4.14: Pérdida de la persona por proximidad

4.3.4. Control en velocidad

Hasta ahora hemos visto la manera de calcular una dirección para realizar el seguimiento de la persona sin chocar con nada. Llegado a este punto, lo que le queda al esquema motriz es modificar la trayectoria del robot variando la velocidad lineal(`v`) y angular(`w`), de modo que éste siga la dirección indicada.

De esta forma, conseguimos un control de la velocidad lineal y angular a través de esas expresiones:

$$v = \sin(\Theta) * (V_{max})$$

$$w = -\cos(\Theta) * (W_{max})$$

siempre que el ángulo de dirección esté entre 0 y π.

$$v = -\sin(\Theta) * (V_{max})$$

$$w = -\cos(\Theta) * (W_{max})$$

siempre que el ángulo de dirección esté entre π y 2π

Para comprender el funcionamiento de estas ecuaciones, supongamos que la trayectoria final deseada viene definida por la dirección de ángulo 20 grados (en el sistema de referencia de la figura 4.12), ya que queremos evitar un obstáculo que se encuentra muy cerca de la parte izquierda del robot (*caso obstáculos en las proximidades*). Lo que necesitamos es que el robot no efectúe un giro muy abierto hacia la derecha. Ésto supondrá que la velocidad lineal del robot estará próxima a cero ($\sin 20 \simeq 0$), mientras que en lo que se refiere a su velocidad angular, estará cerca de la $W_{max}(\cos 20 \simeq 1)$.

De igual modo, si no hubiera obstáculos alrededor, y la persona estuviera de frente al robot (*caso sin obstáculos*), el robot debe ir en línea recta hacia la persona sin que se produzca ningún giro. Por tanto, el valor del ángulo para ir en esa dirección ha de ser igual o próximo a 90 grados, con lo que se consigue que el robot avance a la V_{max} ($\sin 90 = 1$) y apenas gire ($\cos 90 = 0$).

Los aumentos en la velocidad lineal se llevan a cabo acelerando paulatinamente, con el fin de evitar movimientos bruscos, mediante un *filtro de suavidad temporal* en el envío de órdenes de movimiento. El esquema motriz lo primero que realiza es el cálculo de la *velocidad objetivo* según la expresión anterior, y a continuación halla la diferencia entre esa velocidad y la que lleva actualmente. En cada iteración, el esquema `followobject` suma una constante de aceleración Ka a la velocidad actual. La función del *filtro de suavidad temporal* es la de regular el número de iteraciones por segundo que ejecuta `followobject`. De esta forma, si la diferencia entre la velocidad actual y la objetivo resulta ser muy amplia, el número de iteraciones por segundo que efectuará `followobject` será reducido, tratando de alcanzar la velocidad objetivo paulatinamente. A medida que la velocidad actual se aproxima a la objetivo, el número de iteraciones por segundo que ejecuta el esquema `followobject` será mayor, hasta que se produzca:

1 . $velocidadactual = velocidadobjetivo$.

2 . O bien $velocidadactual + Ka \geq velocidadobjetivo$, en cuyo caso velocidad actual = velocidad objetivo.

Cuando se ha alcanzado la velocidad objetivo, el número de iteraciones se mantiene constante, al igual que la velocidad como puede verse en la figura 4.15.

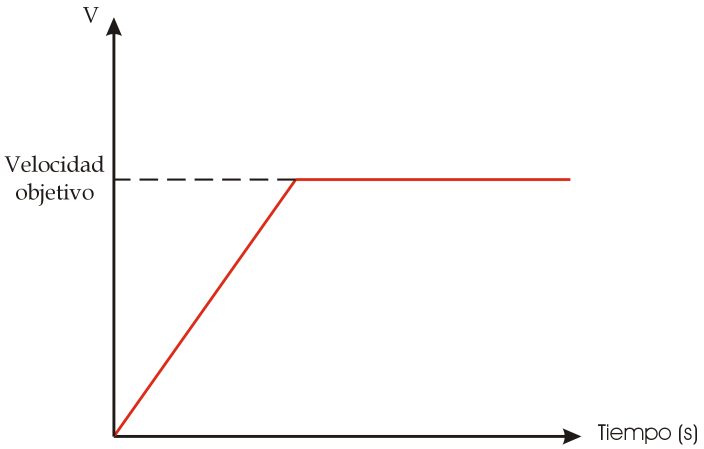

Figura 4.15: Filtro de suavidad temporal

El esquema `followobject` escribe sus valores de velocidad actual sobre dos variables compartidas como son v y w. Una vez hecho ésto, el esquema motriz no se preocupa

de nada más, ya que es la propia plataforma de JDE la encargada de enviar estas órdenes al servidor otos para que las lleve a cabo sobre el propio robot. Ésto lo realiza el esquema de servicio `motors`, como ya se vió en el capítulo 3.

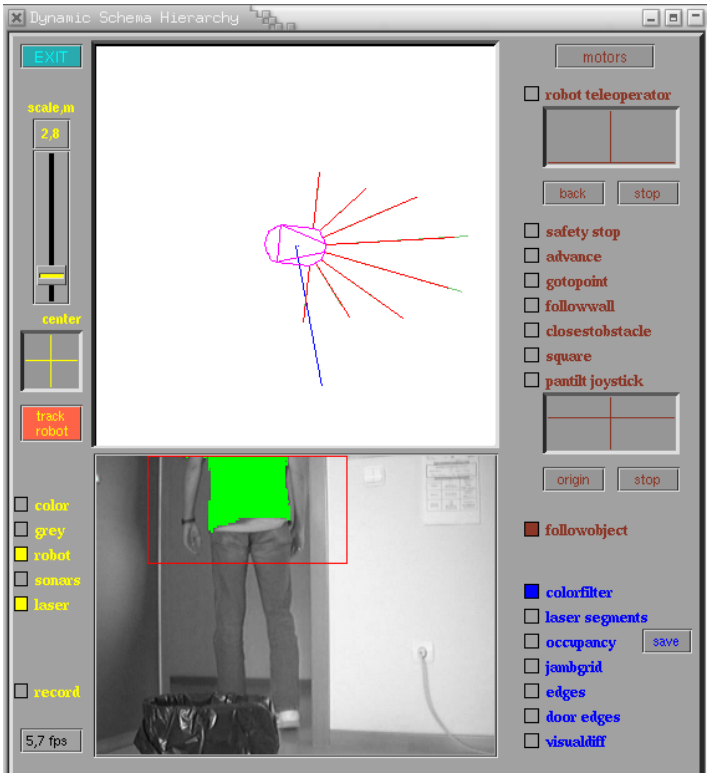

Figura 4.16: Ejemplo del comportamiento mostrado bajo la interfaz JDE

4.3.5. Experimentos

A continuación vamos a describir algunos de los experimentos realizados sobre el robot real, en los que pudimos comprobar como se cumplían los objetivos establecidos para el comportamiento, usando las herramientas de las que disponemos.

En concreto, se realizaron tres experimentos por diferentes entornos para comprobar que, en cada uno de ellos, el comportamiento cumplía los objetivos propuestos.

El primer entorno se corresponde con un pasillo sin obstáculos y con una iluminación constante al que el robot llega tras salir de una habitación. El segundo entorno es un poco más complicado, ya que incorpora una serie de obstáculos que el robot ha de sortear, y entra en una zona de mucha claridad. El tercer experimento se desarrolla en una zona amplia y libre de obstáculos, donde probamos el caso de que la persona se quede parada, parándose también el robot alcanzada una cierta proximidad.

Las velocidades máximas definidas para los experimentos fueron de *600 mm/sec* en la velocidad de tracción y *37 grados/mm* en su velocidad de giro. Por su parte, las distancias que definen la zona de seguridad corresponden a 300 mm para la *distancia*

de seguridad lateral y 1000 mm para la *distancia seguridad frontal*, y para la zona de precaución se definieron 400 mm para la *distancia de precaución lateral* y 900 mm para la *distancia precaución frontal*.

1. El primer experimento se realizó por un pasillo, libre de obstáculos, sobre el que el robot avanza hacia la persona a gran velocidad, tomando el control el (*caso sin obstáculos*). Como puede verse en la figura 4.17, el robot se desplaza de derecha a izquierda del pasillo siguiendo a la persona. Al no detectar nada en su zona de seguridad ni precaución, el robot avanza hacia la persona con velocidad lineal y angular en función del ángulo en el que se encuentre la persona con respecto al robot. En un determinado instante, la persona se encuentra prácticamente alineada con el robot, con lo cual su velocidad lineal y angular dependen de un ángulo muy próximo a 90 grados. Supongamos que el ángulo en el que se encuentra la persona es 101 grados, y al no haber obstáculos en ambas zonas, el robot avanza con velocidad objetivo $v = 588mm/seg$ y $w = 7{,}0grados/mm$. En los primeros experimentos realizados, el esquema motriz no tenía en cuenta la velocidad actual del robot, con lo que se pasaba de $velocidadlineal = 0mm/seg$ a $velocidadlineal = 588mm/seg$ en una sola iteración. Esto provocaba que el robot avanzara con movimientos muy bruscos y discontinuos, con lo que se decidió aplicar el *filtro de suavidad temporal* comentado anteriormente.

Figura 4.17: Ejemplo de seguimiento por un pasillo

De esta manera, en cada iteración sumamos una constante Ka a la velocidad lineal actual, y controlamos el número de iteraciones por segundo. En la tabla 4.3 se aprecia esa aceleración temporal y las velocidades obtenidas en cada instante con una constante Ka de 20 mm/seg.

2. Otro entorno sobre el que se realizaron los experimentos fue un pasillo, con algunos obstáculos alrededor y cambios de iluminación. En esta ocasión toma el control el *caso obstáculos en las proximidades* (figura 4.18). Trabajar bajo este tipo de entorno nos permitió:

Instante	*Velocidad Lineal Actual (mm/seg)*	*Nº iteraciones/segundo*	*T.transcurrido (segundos)*
0	v=0	3.	0.
1	v=60.	3.	1.
2	v=120	3.	2.
3	v=220	5.	3.
4	v=320.	5.	4.
5	v=420.	5.	5.
6	v=588.	10.	6.

Cuadro 4.3: Velocidades obtenidas con el filtro de suavidad temporal

1 Calibrar la *distancia lateral* y *distancia frontal* de ambas zonas, para el caso que el robot tenga que frenar cuando está realizando el seguimiento a gran velocidad. En un principio, las cotas de estas zonas oscilaban entre 100 y 200 mm en sus distancias laterales, y 500 y 700 mm en sus distancias frontales. El resultado era que al robot no le daba tiempo a tomar una decisión de evitación a tiempo y se acercaba demasiado al obstáculo hasta que chocaba. Como solución decidimos aumentar estas cotas hasta 300/400 mm en su *distancia lateral*, y 900/1000 mm en su *distancia frontal*.

2 También nos ayudó a calibrar los valores HS para el filtro HSI. Dado que las primeras pruebas de aplicación de este filtro se realizaron con la iluminación del laboratorio, el filtro estaba diseñado para este tipo de iluminación constante. El trabajar en otro tipo de entorno con grandes cambios de iluminación nos obligó a corregir estos parámetros y ampliar el rango entre $Hmax/Smax$ y $Hmin/Smin$, con el fin de que el comportamiento fuera más robusto.

3 El ampliar el rango de los valores HS supuso un inconveniente, ya que pasaban el filtro pixeles que eran de la camiseta, y pixeles que no lo eran. Además contábamos con una velocidad de análisis de 3 imágenes por segundo, lo que ocasionaba que el comportamiento no cumpliera uno de sus requisitos, la vivacidad. Como resultado de estos experimentos se decidió implementar la *ventana de atención*, de manera que una vez localizada a la persona, sólo se aplicara el filtro sobre una zona en concreto. Con la implementación de la ventana, aumentó la velocidad de análisis de la imagen, llegando a alcanzar el valor de 9 imágenes analizadas por segundo. Además se reducía el número de pixeles que pasaban el filtro que no eran de la camiseta verde, y por tanto, se ajustaba mucho más la posición de la persona en la imagen.

En la figura 4.18, podemos apreciar como el robot sigue la dirección en la que se encuentra la persona, hasta que se encuentra un obstáculo de frente. Es entonces cuando ambas zonas detectan la proximidad del obstáculo y se calcula una

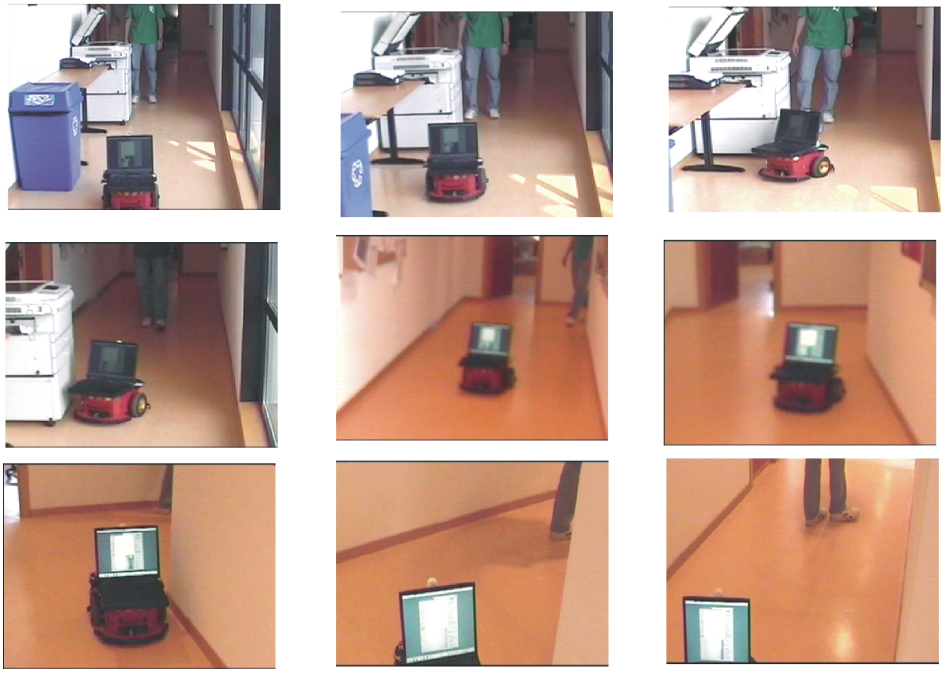

Figura 4.18: Evitación de obstáculos

dirección que lo evite. Esta dirección trata de no perder de vista a la persona, y en cuanto sortea el obstáculo, el seguimiento vuelve ha efectuarse de manera normal.

3. Dentro del mismo escenario de la figura 4.18 toma el control el *caso de búsqueda*. Cuando la persona gira la esquina a la derecha, el robot la pierde de vista. Tras varios experimentos decidimos separar la imagen en diferentes *zonas de búsqueda* que permitieran al robot seguir buscando a la persona si desaparecía de la imagen. En esta ocasión, el robot ve por última vez a la persona en la franja derecha de la imagen, con lo que la búsqueda tomará esa dirección. Los experimentos realizados nos permitieron ajustar estas direcciones, de modo que si el robot pierde de vista a la persona por la derecha, la dirección que sigue viene dada por el ángulo de 45 grados. Ésto permite al robot girar y avanzar en esa dirección. El robot tiene en cuenta en todo momento si le es necesario evitar algún obstáculo, como puede verse en la figura 4.18. Cuando trata de girar a la derecha para buscar a la persona se da cuenta de que puede chocar contra la pared, con lo que efectúa la evitación del obstáculo corrigiendo la trayectoria hasta que puede seguir en la dirección de búsqueda. Cuando la persona aparece de nuevo en su campo visual, el robot continúa el seguimiento de manera normal.

4. El último entorno donde realizamos experimentos tenía iluminación constante y apenas existían obstáculos. En esta ocasión, queríamos que tomara el control el (*caso proximidad*). Para ello la persona se quedaba parada, el robot avanzanba hacia ella a gran velocidad, y cuando estaba a cierta distancia frenaba y se

quedaba parado. En unas primeras pruebas, el criterio que se siguió para indicarle al robot cuando se encontraba la persona cerca era contar el número de pixeles que pasaban el filtro, si este número superaba cierto umbral, la persona estaba muy cerca. Ésto no era del todo fiable, ya que aún teniendo la ventana de atención, se daba el caso en que el filtro fallaba, con lo que el robot no se daba cuenta de que la persona estaba muy cerca y la tomaba por un obstáculo.

Tomamos entonces como referencia cómo estaba instalada la cámara en el robot, la orientación que tenía y el hecho de que durante el seguimiento, el robot trata de mantenerse alineado con la persona. Con ello, y basándonos en el algoritmo de búsqueda, decidimos tomar el criterio de que si el robot había perdido de vista a la persona por la zona superior centrada de la imagen (figura 4.13), entonces era muy probable que fuera porque estaba muy próximo a la persona (figura 4.19).

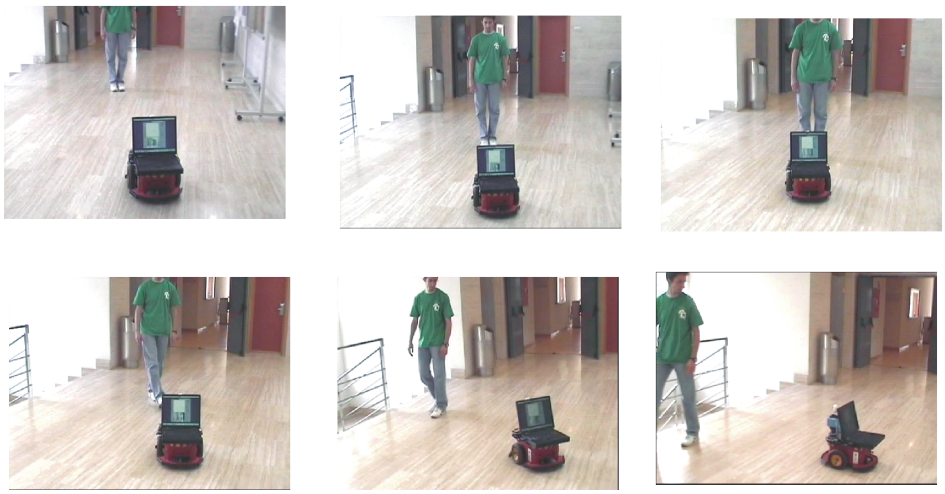

Figura 4.19: Ejemplo de que le robot se detiene si la persona se detiene

Como podemos observar en la figura 4.19, en cuanto la persona se distancia del robot, éste continúa el seguimiento de manera normal.

Capítulo 5

Conclusiones y Mejoras.

En este capítulo analizaremos las conclusiones obtenidas tras generar el comportamiento sigue persona con visión, así como algunas de las dificultades surgidas durante su realización. También veremos posibles líneas futuras que permitan mejorar el comportamiento.

5.1. Conclusiones

La conclusión principal que podemos obtener es haber generado satisfactoriamente un comportamiento que siga a una persona en tiempo real, a la vez que evita obstáculos. El comportamiento se ha generado en el robot Pioneer y usando la plataforma software JDE, que fueron descritos en el capitulo 3.

Además, y tal y como se plantea en el capitulo 2, tras haber desarrollado un comportamiento sigue persona con visión, podemos obtener conclusiones desde dos puntos de vista: conclusiones desde el punto de vista del proyecto y conclusiones desde el punto de vista del comportamiento.

Entre las primeras conclusiones que podemos obtener están las situadas en el contexto del proyecto. Como vimos en el capitulo 4, hemos conseguido diseñar el comportamiento bajo dos esquemas de la plataforma JDE. Uno de ellos perceptivo, que analiza la imagen y extrae información relevante sobre la existencia de la persona y su posición en la imagen. El otro esquema diseñado es un esquema motriz, que se basa en la información extraída de la imagen por el esquema perceptivo para llevar a cabo una determinada acción.

Se han implementado estos dos esquemas de manera que se integran dentro de la plataforma JDE de alto nivel. Estos esquemas utilizan los esquemas de servicio que proporciona la plataforma para la lectura de las medidas sensoriales y el envío de órdenes de movimiento al robot. También utilizan el esquema `guixforms` para visualización.

Para finalizar las conclusiones en el contexto del proyecto se encuentran los experimentos realizados, donde pudimos ver que el comportamiento funcionaba

correctamente en un robot real. Estos experimentos nos permitieron realizar varios ajustes. El primero sobre el filtro de color para la camiseta verde, cuyos valores HS fueon ajustados tras probar el comportamiento en diferentes entornos con diferentes cambios de iluminación. El segundo nos permitió ajustar la *distancia lateral* y *distancia frontal* de las zonas de seguridad y precaución, de modo que el robot diferencia entre cuando tiene que evitar un obtáculo y cuando no. También los experimentos nos permitieron ajustar las zonas de búsqueda en la imagen, para que el robot continuara buscando durante un tiempo a la persona si la perdía de vista, o frenara porque estaba próximo a ella.

Dentro del contexto del comportamiento también considerábamos dos subobjetivos, como vimos en el capitulo 2. Entre ambos se establecía un compromiso *seguridad-seguimiento* de tal forma que el robot fuese capaz de seguir a la persona sin chocar con nada. Ambos objetivos se han combinado en diferentes casos relevantes, de manera que dependiendo del caso en el que se encuentre el robot, se aplica una determinada regla de control.

Para implementar cada uno de estos casos ha sido necesario apoyarse en la arquitectura de control JDE, plataforma sobre la que hemos integrado nuestros programas. En concreto dos esquemas concurrentes, uno de ellos perceptivo, encargado de localizar a la persona en la imagen, y el otro motriz, encargado de enviar las órdenes de movimiento al robot. El primero de ellos, `colorfilter`, corresponde al esquema que analiza la imagen para identificar y localizar a la persona. Ésta información sobre la dirección en la que se encuentra el objetivo a seguir es aprovechada por otro esquema, que corresponde al esquema motriz `followobject`, el cual envía órdenes de movimiento a los motores del robot para dirigirle hacia la persona.

En lo relacionado con el esquema perceptivo, se ha logrado que la identificación de la persona en la imagen se haga de manera robusta frente a cambios de iluminación, aplicando un filtro HSI sobre la imagen. De esta manera se cumple el requisito propuesto en el capitulo 2 y que hacía referencia a la robustez. Inicialmente este objetivo se trató de resolver con un filtro sobre el espacio RGB, pero se comprobó que un filtro bajo esta clase de espacio era menos robusto que el filtro HSI. Al implementar este filtro contábamos con la dificultad del ajuste automático de color que realiza la cámara (autoiris), la cual variaba el valor de la componente H de un determinado objeto que captara de repente. Este valor no se vuelve estable hasta que ese objeto permanece un tiempo en el campo visual de la cámara. Aun así, el filtro HSI sigue siendo lo suficientemente robusto como para permitir localizar a la persona, a pesar de esta dificultad añadida.

Se ha conseguido llegar a analizar la imagen a un ritmo de 9 fps, tras comprobar que una vez localizada la persona no es necesario seguir analizando toda la imagen,

sino sólo una zona respecto a esa posición (*ventana de atención*). Ésto aumenta la vivacidad del comportamiento, con lo que se cumple otro de los requisitos planteados en el capítulo 2. Con todo ello se ha llegado a la conclusión de lo necesario que resulta la robustez y vivacidad a la hora de analizar las imágenes que el robot captura a través de su cámara, ya que aumentan la calidad del comportamiento. A esta conclusión hemos llegado tras observar la importancia de estos dos requisitos en otros comportamientos como el *comportamiento sigue pelota en un robot con visión local* [SanMartín02], el *comportamiento sigue pared* [Gomez02] o el *comportamiento sigue pelota con visión cenital* [Martinez03].

En lo que se refiere al esquema motor, se consigue que el comportamiento sigue persona se ejecute en tiempo real y sin que se produzcan movimientos bruscos ni tirones. Hemos llegado a la conclusión, tras realizar varios experimentos, de que el número de iteraciones de control es un factor crítico en el comportamiento, ya que realizamos un control en velocidad.

En lo referente a la evitación de obstáculos nos dimos cuenta de lo importante que resultó la creación de dos zonas para detectarlos. Dependiendo de si alguna de estas zonas detectaba la presencia de obstáculos, se tomaba una decisión de actuación u otra, lo que permite discriminar de modo muy preciso cuando hay que *"hacer caso"* a un obstáculo próximo. Si ninguna de las dos zonas detectaban obstáculos, significaba que nada impedía realizar el seguimiento de manera normal, lo que mejora la calidad del comportamiento ya que se consigue un seguimiento más real y continuo. Si alguna de las dos zonas detectaba obstáculos, nos permitía discriminar entre reducir la velocidad, o en un caso extremo, sortear el obstáculo. En esta última situación, la aplicación en la evitación de obstáculos del método basado en campos de fuerzas virtuales (`VFF`), nos permitió generar una dirección alejada del obstáculo y sin desviarse demasiado del objetivo a seguir.

Si la persona desaparece de la imagen, comprobamos que podíamos considerar varios casos dependiendo de la última dirección en la que la vió el robot. Con ésto el robot realizaba una búsqueda en esa dirección durante un tiempo definido. Una vez la persona aparecía en la imagen, el robot continuaba el seguimiento de manera normal. Ésto supone un avance con respecto a comportamientos de seguimiento como el *comportamiento sigue pelota en un robot con visión local* [SanMartín02], donde si se perdía a la pelota el robot se paraba y no trataba de buscarla.

Aprovechamos el caso en el que la persona desaparece de la imagen por una zona superior centrada, para considerar que estaba próxima al robot, con lo que debía pararse.

En resumen, hemos llegado a la conclusión de lo importante que es la robustez y vivacidad en la percepción, ya que ayuda al comportamiento a tomar decisiones en la

parte de control en menor tiempo, y con ello conseguimos un seguimiento en tiempo real.

5.2. Líneas Futuras

La calidad del comportamiento sigue persona conseguido es satisfactorio, aun así, hay algunos puntos por los cuales se podría mejorar su calidad.

1. La primera mejora está relacionada con la percepción, de tal manera, que podría incorporarse un filtro aún mejor del que actualmente se ha implementado, como un filtro de Kalman. Con este tipo de filtro se puede predecir hacia dónde se va a dirigir la persona relacionando las observaciones realizadas en el instante actual y en el instante anterior. En el comportamiento sigue persona desarrollado en este proyecto estas observaciones son independientes, salvo en la ventana de atención. A sí mismo, se podría realizar un filtrado por bordes en lugar de por color.

2. En la actualidad, la cámara ocupa una posición fija a bordo del robot, con lo que una posible mejora sería poder utilizar una plataforma hardware, como una *pantilt* o cuello mecánico, para permitir mover la cámara sin necesidad de mover el robot [Calvo04].

3. Otra mejora se centra en el aspecto de la detección de obstáculos, ya que actualmente se realiza con los sensores de ultrasonidos del robot Pioneer, los cuales son bastante imprecisos, debido a varias incertidumbres en sus medidas como vimos en el capitulo 3. El uso de un sensor capaz de analizar el entorno con mucha más precisión, como el sensor láser, ayudaría a mejorar el seguimiento y diseñar una evitación de obstáculos más fiable que la actual. El sensor láser aumentaría la calidad del comportamiento, permitiéndo que el robot pasara de manera más segura por lugares estrechos.

4. Se ayuda a la identificación de la persona, ya que va vestida con una camiseta de color poco frecuente. Como mejora, la identificación se podría realizar sin este tipo de ayuda, realizándose una localización por reconocimiento de caras.

5. El comportamiento sigue persona sólo usa la cámara para detectar la presencia del objetivo a seguir. Se podría realizar una *detección multimodal* de la persona, de tal forma que se combinen el sensor cámara y el sensor láser para percibir su presencia.

Bibliografía

[Cañas03] CAÑAS, J.M.: *"Jerarquía Dinámica de Esquemas para la generación de comportamiento autónomo."*, Tesis Doctoral, Universidad Politécnica de Madrid, 2003.

[Cañas03b] CAÑAS, J.M., MATELLÁN, V.: *"Manual de programación para el robot Pioneer."*, Informe Técnico del Gsyc, 2003.

[Cañas04] CAÑAS, J.M.: *"Manual de programación de robots con JDE."*, Informe Técnico del Gsyc, 2004.

[Calvo04] CALVO, R.: *"Comportamiento sigue persona con visión direccional."*, Proyecto Fin de Carrera, Universidad Rey Juan Carlos, 2004.

[Martinez03] MARTINEZ DE LA CASA, M.: *"Comportamiento sigue pelota con visión cenital."*, Proyecto Fin de Carrera, Universidad Rey Juan Carlos, 2003.

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

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

[Gomez02] GÓMEZ, V.: *"Comportamiento sigue pared."*, Proyecto Fin de Carrera, Universidad Rey Juan Carlos, 2002.

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

[Borenstenin89] BORENSTENIN, J., KOREN, Y.: *"Real-time obstacle avoidace for fast mobile robots*, IEEE Journal of Robotics and Automation,1989.