UNIVERSIDAD REY JUAN CARLOS

Ingeniería Técnica en Informática de Sistemas

Escuela Superior de Ciencias Experimentales y Tecnología

Curso académico 2002-2003

Proyecto Fin de Carrera

Filtro de color configurable

Tutor: José María Cañas Plaza

Autor: Alfonso Matute Baena

Junio 2.003

A mis amigos.

Agradecimientos

Quiero agradecer al Grupo de Robótica de la URJC y en especial a José María Cañas su ayuda y paciencia en el desarrollo de este proyecto.

Índice general

Índice de figuras

Capítulo 1

Introducción

Es difícil situar el nacimiento de la robótica ya que en varios momentos de la historia se han creado máquinas con diferentes tecnologías que se podrían considerar precursores de los robots.

El objetivo principal de la robótica ha sido históricamente construir máquinas que pudieran realizar ciertas actividades difíciles o incluso imposibles para el hombre. Esta curiosidad del hombre por crear inteligencia artificial se ha reflejado en la ciencia ficción, donde destacan las obras de Isaac Asimov (1939) y Karel Kapec (1917), quien dio origen a la palabra robot, que proviene del checoslovaco y significa "trabajador". En su evolución incluso se han denominado con ese nombre aparatos destinados al entretenimiento, como los autómatas humanoides fabricados con mecanismos de relojería durante los siglos XVII y XVIII.

La primera aplicación real de estos automatismos surgió durante la revolución industrial, en el sector textil. Fueron los telares mecánicos las primeras máquinas diseñadas para realizar el trabajo con mayor rapidez y eficiencia de lo que hasta entonces realizaban los hombres. El desarrollo de la robótica se ha basado principalmente en aplicaciones industriales tales como cadenas de montaje y dispositivos que necesitaban gran precisión y rapidez de movimientos, por ejemplo, robots manipuladores.

Unido al desarrollo de la informática y la electrónica, se ha alcanzado en la actualidad tal nivel que los robots están adquiriendo posibilidades comerciales en el ámbito doméstico y de ocio, lo que impulsa su crecimiento. Por ejemplo el robot Roombavac [1], que consiste en una aspiradora robótica. También destacan las mascotas robóticas como el robot Aibo [2]. Otro ejemplo es un robot capaz de guiar un grupo de visitantes por un museo, sabiendo en todo momento dónde se encuentra. Se trata del robot Minerva, desarrollado en Carnegie Mellon [3].

La robótica ha impulsado la investigación en diferentes disciplinas. Dos de ellas son la inteligencia artificial y la visión computacional. Dentro de la inteligencia arti-

[1] http://www.roombavac.com
[2] http://www.aibo.com
[3] http://www-2.cs.cmu.edu/ minerva

ficial se han realizado avances en campos como el cálculo de trayectorias, el diseño de mapas, etc. La disciplina de la inteligencia artificial investiga maneras de emular, con computadoras electrónicas el procesamiento de la información de forma similar a los humanos. En el campo de la visión artificial también se están llevando a cabo importantes investigaciones.

1.1. RoboCup

Hace algunos años, el profesor Alan Mackworth[4] planteó por primera vez la idea de desarrollar jugadores robóticos de fútbol, como impulsor de la investigación en robótica. Incluso planteó el reto de crear un equipo de robots capaces de batir en el futuro a un equipo de jugadores humanos.

Independientemente, un grupo de investigadores japoneses organizó un taller sobre Grandes Retos de La Inteligencia Artificial en Octubre del 1992 en Tokio, discutiendo temas relacionados a la IA. Este taller condujo a discusiones serias sobre el uso del fútbol para promover la ciencia y tecnología.

Como resultado de estos estudios, junto con otros relacionados con la viabilidad de la tecnología, un estudio del impacto social y de la viabilidad financiera se planteó la creación de la primera competición de fútbol robótico. Esta se llamó J-League (Liga Profesional Japonesa de Fútbol). Pero un mes después, ante las peticiones de investigadores de fuera de Japón, la competición fue ampliada a participantes de todo el mundo. Esta nueva competición fue llamada Robot World Cup Iniciative[5], cuyos primeros participantes provenían de las universidades de Osaka y Carnegie Mellon, que ya llevaban tiempo investigando en robótica. El encargado principal de llevar esto a cabo fue Hiroaki Kitano. En el año 1996 se realizó la Pre-RoboCup-96, que sirvió de ensayo para localizar problemas potenciales asociados a la organización de una RoboCup a gran escala. La primera RoboCup se organizó en el año 1997, con grán éxito de participación: 40 equipos y alrededor de 5.000 espectadores.

La RoboCup se organiza en diferentes categorías. Las diferencias entre ellas están en el tipo de robots que se utilizan. Las ligas existentes son:

- Liga de simulación. Se utilizan dos equipos de 11 robots autónomos simulados mediante clientes software, que funcionan sobre un servidor que genera las situaciones de juego. Los participantes pueden crear su propio cliente (ó equipo) para esta aplicación y participar en una competición en la que se eliminan los problemas técnicos que conlleva utilizar robots reales. El `Robocup Soccer Simulator` es una herramienta

[4] (University of British Columbia, Canada), en un documento titulado "On Seeing Robots" presentado en 1992 y publicado más adelante en el libro "Computer Vision: System, Theory, and Applications".

[5] http://www.robocup.com

educativa y de desarrollo para sistemas multi-agente e inteligencia artificial.

- Liga de pequeños robots. Deben caber en un cilindro de 18 cm de diámetro y una altura máxima de 15 cm. Juegan sobre un campo de 2.4 x 2.9 m, similar a un tablero de ping-pong, de color verde uniforme y con lineas blancas que delimitan el campo, las áreas de las porterías y el centro del campo. Utilizan como balón una pelota de golf de color naranja. Los robots deben poderse distinguir entre ellos con una etiqueta de determinado color para distinguir los robots de cada equipo. Además, pueden llevar otras etiquetas, a juicio de los participantes, que sirvan para tareas auxiliares. Se permite utilizar un sistema de visión global, con una cámara cenital, situada sobre el campo. Como mínimo puede participar 1 robot por equipo y como máximo 5, aunque éstos pueden ser sustituídos tantas veces como se quiera.

Figura 1.1: Categoría de robots pequeños

- Liga de robots medianos. Tanto los robots como el campo y la pelota son de mayor tamaño. Utilizan visión local en lugar de global y la toma de decisiones también se realiza localmente, en lugar de estar conectados a un equipo exterior.

- Liga de humanoides (a partir del 2002). De momento sólo se organiza en la categoría de simulación (figura 1.2).

Figura 1.2: Humanoides de mayor tamaño

- Liga Sony Legged (Patrocinada por Sony). Sólo se utilizan robots Aibo de Sony (ver figura 1.3).

Figura 1.3: Mascotas de la marca Sony

Además de las competiciones de fútbol, se realizan otras como la RoboCupRescue y la RoboCupJunior, de momento ambas en la modalidad de simulación.

Aparte de la RoboCup, existen otras competiciones en diferentes universidades con objetivos pedagógicos y de investigación. Por ejemplo, se participa en modalidades de lucha, similar al sumo y carreras por circuitos. Este tipo de competiciones está creando cada año una mayor expectación entre el público y cabe destacar su utilidad para el aprendizaje y desarrollo de la robótica, además de otras técnicas necesarias para poner en funcionamiento un equipo de robots en un escenario dinámico y competitivo..

1.2. Grupo de Robótica de la URJC

El Grupo de Robótica está compuesto por alumnos y profesores de la Universidad Rey Juan Carlos interesados en llevar a cabo investigaciones sobre la generación de comportamiento artificial en robots. Para ello, se hacen estudios sobre lógica borrosa, visión artificial, estimación, elaboración de mapas, teoría de control, arquitecturas de control, etc.

Estas tareas se orientan en dos campos: la creación de un equipo para la participación en la RoboCup y la generacion de comportamientos autónomos en robots de interiores. Para este trabajo se utilizan diferentes modelos de robots, con diferentes capacidades. Por ejemplo, durante el año 2002 se adquirió un modelo de Pioneer, de mayor tamaño, que cuenta con una corona de sensores sónar. La ventaja de este robot es que se le puede añadir un PC portátil, ganando así en capacidad de proceso (figura 1.4)

Durante el año 2000 el grupo planteó la creación de un equipo de robots para participar en la RoboCup dentro de la categoría de tamaño pequeño. Debido a la dificultad de construir un robot propio para la competición se optó por adquirir un

Figura 1.4: Robot Pioneer con PC portátil y cámara para visión

robot comercial de los existentes en el mercado. El robot elegido por el equipo de la Universidad fue el EyeBot (figura 1.5), debido a su aceptable capacidad de proceso. Además, cumple los requisitos para participar en la Robocup, dentro de la categoría de robots de menos de 18 cm de diámetro. Sobre este robot se está trabajando en el desarrollo de los diferentes comportamientos necesarios para organizar un equipo capaz de participar en la RoboCup. También cuenta con una cámara en color incorporada, lo que ha permitido realizar un sistema de seguimiento de la pelota por visión, que filtra el color de la imagen tomada, localizando el de la pelota [SanMartín02].

Otra aplicación que se le ha dado a la cámara del EyeBot es un comportamiento "sigue pared", que procesa las imágenes tomadas por la cámara y mantiene la trayectoria del robot paralela a una pared, salvando las esquinas [Gómez02]. Este mismo comportamiento se ha realizado con los sensores de infrarrojos que incorpora el EyeBot. También está equipado con una estación emisora y receptora de radio. Esto ha sido utilizado para realizar aplicaciones de comunicación entre los robots y un PC.

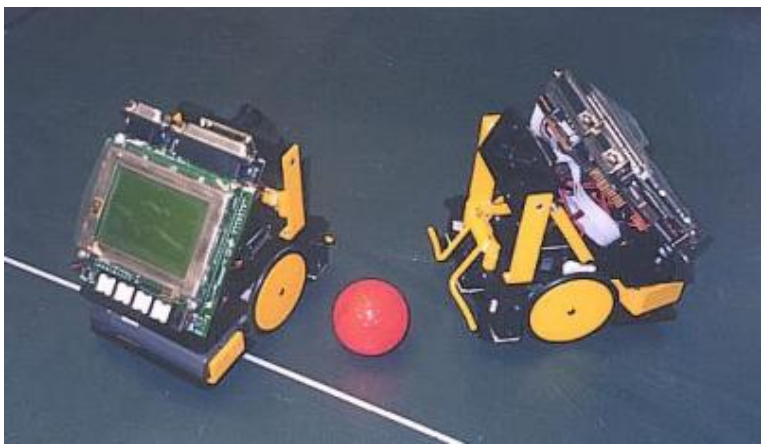

Figura 1.5: Robots modelo Eyebot

Puesto que para participar en la RoboCup, además de los robots, se puede utilizar

un equipo informático que controle la situación del juego en cada momento y les suministre la información necesaria para que puedan actuar, se abren nuevos campos de investigación. Por ejemplo, la visión computacional es necesaria para participar en la categoría de robots pequeños, ya que se utiliza un sistema de visión global, conectado al equipo informático anteriormente mencionado. Aquí surgen problemas que también deben ser resueltos: soporte de la cámara elegida y captura de la imagen en un formato adecuado para su tratamiento. En dicho tratamiento se utilizan técnicas de filtrado en diferentes espacios de color. También, con el fin de aumentar el rendimiento de la aplicación, se utilizan métodos de predicción, muy usados en visión artificial. Como el fin último de la visión global en el equipo RoboCup es la localización de los jugadores y la pelota con la mayor precisión posible, hay que recurrir a la segmentación de la imagen.

1.3. Filtro de color configurable

El presente proyecto pretende resolver los diferentes problemas que presenta la captura y tratamiento de dicha imágen en el escenario de la RoboCup. Por tanto, se trata de conseguir que el PC, a partir de la imagen obtenida con la cámara situada sobre el campo, sea capaz de reconocer los colores de las etiquetas que llevan los robots y el de la pelota. Para ello utilizaremos diferentes filtros de color, mediante los cuales extraeremos de la imagen sólo las zonas cuyo color nos interese (pelota, etiquetas de los robots, bordes del campo, etc.). Esto además nos servirá para comprobar la eficacia de cada filtro en diferentes condiciones de iluminacion. El principal problema son precisamente los cambios en esta iluminación. El cerebro humano apenas se da cuenta de lo diferente que es el color del mismo objeto, captado por el ojo ante diferentes condiciones de iluminación. Pero la cámara no tiene esa capacidad de adaptación y por tanto, este sistema debe tener en cuenta esos cambios y adaptarse de la mejor manera posible. De tal forma que siga reconociendo un mismo objeto aunque cambien las condiciones de iluminación y por tanto, el color de ese objeto que es captado por la cámara.

Además de pasar el filtro, necesitaremos agrupar las zonas con pixels de cierto color para hallar las coordenadas del objeto que representa. Esto debería coincidir con la posición de los robots y demás elementos en cada momento. Así se consigue extraer de una imagen captada por la cámara la posición y orientación de determinados objetos que previamente hemos elegido mediante su color característico.

Posiblemente, el factor más importante y también más difícil de desarrollar es un método para que el usuario humano pueda "decirle" al sistema qué objetos quiere que localice en la imagen. Para ello será necesario configurarlo de forma que sólo pasen el filtro aquellos colores que el usuario quiera. Por tanto, ademas de el sistema explicado

anteriormente, necesitamos una aplicación que permita seleccionar los colores de los objetos que aparecen en la imagen.

La presente memoria se desarrolla en cinco capítulos en los que se aborda la implementación de un filtro de color configurable. En el segundo capítulo se explican con mayor profundidad los objetivos del proyecto y en el tercero se dan detalles sobre las librerías y herramientas utilizados para la captura y la interfaz gráfica. La implementación que hemos desarrollado ocupa el cuarto capítulo. Las conclusiones del presente proyecto y los trabajos futuros corresponden al quinto y último capítulo.

Capítulo 2

Objetivos

En este segundo capítulo se describen los objetivos que persigue la realización de este proyecto, así como los requisitos de partida que es necesario satisfacer.

Recordemos que el uso típico e inmediato para el que vamos a emplear este proyecto es que forme parte de un equipo para participar en la RoboCup. Debido a las características del juego desarrollado en la RoboCup, explicadas en el capítulo anterior, es conveniente que los robots conozcan en todo momento su posición en el campo, así como la de la pelota. Puesto que los odómetros con los que cuenta cada robot son poco fiables, los equipos de la liga de robots pequeños, utilizan un sistema de visión global por computador que utiliza una cámara situada sobre el campo.

Este proyecto consiste en la creación de una serie de herramientas (librerías, aplicaciones, etc.) que nos sirvan para trabajar en el campo de la visión artificial, principalmente en el filtrado de colores con el fin de localizar objetos en una imagen. El motivo que nos ha llevado a realizar este proyecto es la necesidad de resolver el problema de la localización de los robots utilizando una cámara cenital y un PC. Necesitábamos un sistema robusto y estable, capaz de capturar, a partir de las imágenes de la cámara, las coordenadas dentro de la imagen, de los robots y la pelota. Pero en el desarrollo se ha mantenido la idea de dejar su uso abierto a otras actividades.

De este problema principal surgen varios subproblemas:

- La captura de la señal de video procedente de la cámara.

- La implementación de un filtro basado en color que se pueda aplicar a las imágenes consecutivas que forman la señal de video. Es de gran importancia que el filtro sea resistente a los cambios de iluminación.

- La creación de una herramienta que permita al usuario configurar el filtro según sus necesidades.

- La implementación de un proceso de segmentación de la imagen, que permita interpretar las areas que superen el filtro anterior como objetos y extraer sus coordenadas en la imagen.

Esta sería una descripción breve de las partes necesarias para este proyecto. Además podemos implementar una funcionalidad de seguimiento de objetos, que puede ser útil

para mejorar el comportamiento de los robots. Estos, en lugar de dirigirse al punto donde se encuentra la pelota en cada instante, pueden calcular su trayectoria para interceptar la pelota en el lugar donde se encontrará situada en instantes posteriores. Además, mediante la visión global, los robots pueden mantener una estrategia de juego, definiendo las zonas del campo en las que jugará cada robot, al igual que se hace en el fútbol real.

Otra ventaja de este proyecto es que el diseño deja su uso abierto a otros escenarios, como por ejemplo, el seguimiento de personas, caracterizadas por ropa de determinado color. Este sistema realiza un tratamiento de la imagen, en el que se utilizarán diferentes filtros basados en distintos espacios de color. Por ello servirá para realizar ensayos sobre cuál de ellos es más eficaz en esta labor. Con esto se pretende que el proyecto sea una herramienta para vision por computador bastante flexible, que pueda ser utilizada en otros proyectos.

2.1. Objetivos

El objetivo principal se ha articulado en cuatro partes concretas: biblioteca de visión, herramienta de configuración, biblioteca de segmentación y cliente típico.

Biblioteca de filtros de color

Desarrollaremos una biblioteca de visión que conste de filtros de color en varios espacios de color. Este filtro podrá ser de diferentes tipos. Nos interesa implemetar varios tipos de filtro para comparar sus capacidades y utilizar el que más nos convenga en cada situación. El filtro de color se podrá configurar para que trabaje distinguiendo colores definidos en varios espacios simultaneamente.

La entrada que recibirá el filtro será la imagen de la cámara, previamente convertida al formato que nos interese. Sólo procesará una imagen a la vez, pero la biblioteca deberá permitir ejecutar el filtro sobre imágenes consecutivas, de forma que podamos utilizarlo para tratar el flujo de imágenes capturada por la cámara de video. También nos interesa poder suministrarle al filtro una imagen proveniente de otras fuentes, como una almacenada en un fichero.

Los filtros de la biblioteca serán configurables a través de fichero, es decir, utilizarán los parámetros almacenados en un fichero de configuración. Según estos parámetros, se ajustará el filtro a aquellas características definidas por el usuario. Tras procesarla, el filtro devolverá otra imagen del mismo tamaño y en el mismo formato, incluyendo sólo aquellos colores que hayan superado el filtro.

Herramienta de calibración

Mediante una herramienta de calibración para el filtro de color, podremos realizar los cambios en el fichero de configuración de una forma más cómoda. Para configurar correctamente el filtro necesitamos obtener cierta información característica de cada color que queramos filtrar en la imagen. Esta herramienta deberá facilitar la información de forma sencilla e intuitiva. La idea principal de esta herramienta es que el usuario no tenga por qué conocer los valores numéricos que representan a los colores, sino que pueda ver los colores directamente y seleccionar aquellos que desee filtrar. Esta aplicación mostrará al usuario tanto la imagen real como la filtrada, para comprobar la eficiencia de los cambios que realiza en el filtro.

Además, dada la necesidad de hacer trabajar el sistema en diferentes ubicaciones, esta calibración será bastante frecuente y por lo tanto, debemos hacerla lo más cómoda y rápida posible. Puesto que en principio, sólo necesitaremos utilizar esta aplicación antes de comenzar cada juego, nos interesará hacerla independiete del uso del filtro. Es decir, no será necesaria una vez que hayamos configurado el filtro. De este modo aprovecharemos los recursos del PC exclusivamente en el control del juego, ya que la visualización de las imágenes y de la interfaz gráfica consume mucho tiempo de proceso, que será necesario aprovechar para el funcionamiento de la biblioteca de visión.

Biblioteca de segmentación

La segmentación consiste en localizar en la imagen determinadas áreas del mismo color, es decir, buscar zonas de determinado color uniforme e interpretarlas como objetos. Nuestra librería incluirá funciones que permitan realizar este procesado. También deberá hacer una estimación de la posición de cada área localizada. Estos datos nos permiten interpretar dichas zonas como objetos dentro de la imagen del campo y hacer un seguimiento de los robots y la pelota en todo momento. Interesa que pueda hacer esto con varios colores y varios tipos de filtro simultáneamente. Como entrada recibirá la imagen resultante del filtro y ciertos parámetros de configuración.

Cliente típico

Para facilitar el uso de las librerías desarrolladas, implementaremos, a modo de ejemplo, una aplicación que las utilice. Además, servirá como esqueleto para que otros usuarios puedan utilizarlo en otras aplicaciones insertando el código del cliente típico en su aplicación, haciendo más cómodo su uso. Realizará las siguientes acciones: lectura de la imagen proveniente de la captura, filtrado de la imagen según los parámetros del fichero de configuracion, creación de una nueva imagen que sólo muestre la información que no haya sido descartada por el filtro y segmentación de cada uno de los elementos que aparezcan en la imagen resultante del procesado con el filtro de color.

2.2. Requisitos

Además de alcanzar los objetivos mencionados en el apartado anterior, necesitamos cumplir unos requisitos que hagan el sistema válido para participar en la Robocup.

Robustez frente a variaciones de iluminación

El principal requisito de nuestro filtro de color es la robustez ante cambios en la iluminación del escenario capturado por la cámara. Es decir, nos interesa suavizar estas variaciones de forma que la imagen permanezca lo más estable posible. En caso contrario, se podría perder la posición de algún robot si éste entra en una zona de distinta iluminación. El robot podría no saber salir de dicha zona al no conocer su propia situación. Las sombras (tanto las estáticas como las producidas por personas que se mueven cerca del campo), los reflejos de las luces y el diferente grado de iluminación de cada zona del campo supone un problema para reconocer los colores característicos de los elementos importantes en la imagen, ya que la cámara capta un color diferente al aumentar su intensidad por efecto de la iluminación. Por tanto, esto también debe ser controlado. Mediante este filtro podremos explorar diferentes espacios de color para seleccionar el/los que sean más apropiados para nuestros fines, así como comprobar su invarianza respecto a cambios de iluminación.

Vivacidad

Debido a la vivacidad del juego de los robots sobre el campo y el movimiento de la pelota, es necesario que el tiempo de proceso de cada imagen sea lo más reducido posible. El ideal sería alcanzar un rendimiento cercano a la velocidad de captura de video, pero sin utilizar, de momento, hardware específico. Por tanto, un requisito de la implementación será su máxima sencillez, evitando bucles innecesarios.

Flexibilidad

Aunque este proyecto se ha desarrollado para una aplicación concreta en el equipo Robocup, puede ser utilizado para otros trabajos en el campo de la visión por computador. Para aprovechar esta flexibilidad, la interfaz debe ser sencilla e intuitiva. De este modo, cualquier persona ajena al entorno donde se ha desarrollado podrá utilizarla con otros fines. Por el mismo motivo se ha añadido la funcionalidad de realizar el filtrado de una imagen almacenada en un archivo en lugar de la captada por la cámara, además de para poder calibrar desde una imagen en fichero.

Nos interesa dar soporte a diferentes tipos de cámaras para que, una vez terminado el proyecto, podamos probar varios modelos y decidir cuál se ajusta mejor a nuestras necesidades. Además debemos poder utilizar distinto hardware en este proyecto, no solo para que podamos probar con diferentes cámaras en nuestro equipo de la Robocup,

sino que quien quiera utilizar este filtro de color con diferentes fines al nuestro, pueda, del mismo modo, elegir el hardware que necesite y éste sea soportado por la aplicación.

Capítulo 3

Herramientas

El desarrollo de este proyecto no habría sido posible sin la utilización de algunas librerías y utilidades. Recordemos que la aplicación inmediata de este proyecto es filtrar y analizar la imagen del campo de juego de la RoboCup mediante una cámara de video conectada al PC. Para empezar, era necesario solventar el problema de la captura de esta imagen desde la cámara. Para ello se ha utilizado el interfaz VideoForLinux.

Puesto que la herramienta de configuración del filtro debe resultar cómoda en su manejo, la mejor solución consiste en un interfaz gráfico que permita al usuario la configuración visual del filtro. En Linux, el sistema de ventanas más extendido es X-Windows. Existen varias librerías de libre distribución que trabajan sobre X-Windows y que facilitan el desarrollo de dichos interfaces, como Gtk+ ó Qt. En esta ocasión se ha elegido la librería Xforms.

3.1. Interfaz VideoForLinux

VideoForLinux (también Video4Linux ó v4l) es un interfaz de programación diseñado para dar soporte en linux a muchas de las capturadoras de video y TV que existen actualmente en el mercado, así como a cámaras USB y paralelo. También está diseñado como interfaz para dispositivos de radio y de teletexto. Su objetivo principal consiste en unificar la interfaz para hardware muy diverso de captura de imágenes, de manera que la misma aplicación funcione sobre hardware diferente sin apenas retoques de código. Esto es una gran ventaja para este proyecto ya que nos permite dejar abierto su uso a diferentes dispositivos sin que el usuario que lo utilice tenga que retocar el código. De hecho, para la realización de este proyecto se ha dispuesto de una tarjeta capturadora BT878 a la vez que se ha utilizado una webcam USB.

El siguiente esquema aclara dónde se sitúa VideoForLinux dentro del sistema:

VideoForLinux asigna a cada tarjeta de vídeo una serie de entradas en el directorio /dev, que es en el que se situan los distintos dispositivos en linux. En el caso del vídeo asigna una entrada /dev/videoXX por cada dispositivo capaz de manejar señales de

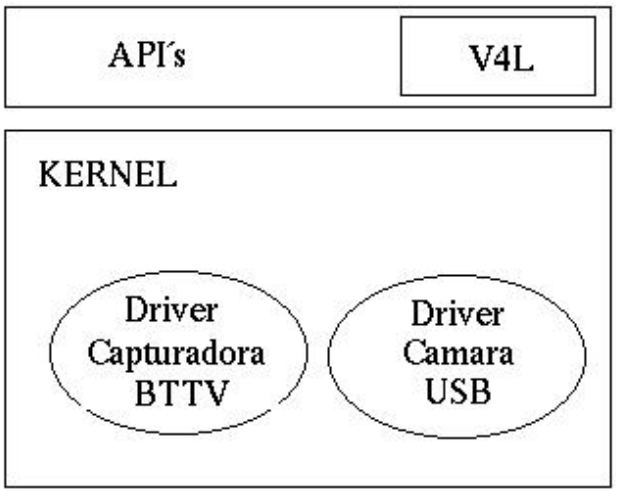

Figura 3.1: Situación del API VideoForLinux en el sistema LINUX

vídeo (con o sin audio asociado). Para las señales de radio y teletexto haría lo mismo con /dev/radioXX y /dev/vtxXX.

Cada dispositivo de vídeo tiene asociado uno o varios canales, que a su vez pueden tener o no dispositivos de sintonización. La interfaz VideoForLinux permite a los programas elegir cual de ellos se selecciona como fuente de la señal de video.

En el sistema Linux/Unix, las aplicaciones interactúan con el módulo videodev, que es el que ofrece la interfaz de VideoForLinux. El uso típico de VideoForLinux para capturar imágenes sigue una serie de pasos:

1. Apertura de dispositivos de video.
2. Identificar prestaciones de cada dispositivo.
3. Seleccionar el modo de funcionamiento (formatos, tamaño, etc).
4. Lectura de imágenes:

a) por frames, mediante la función read.

b) por captura continua con mmap.

5. Cierre mediante la llamada al sistema close().

Mediante llamadas ioctl(), VideoForLinux nos permite tener acceso al control de los dispositivos de video e interactuar con la tarjeta capturadora de video. El uso típico de V4L desde un programa consta de una serie de pasos, introducidos en el párrafo anterior:

La apertura del dispositivo de video se realiza mediante la llamada al sistema `open()`, indicándole como parámetro el nombre que el sistema asigna a dicho dispositivo: /dev/video.

Lo segundo que debemos hacer para manejar un dispositivo de vídeo desde el sistema operativo del PC es saber qué funcionalidades posee y cómo las podemos manejar. Por

ejemplo, podemos comprobar: el nombre de la capturadora, sus capacidades (si tiene teletexto, si permite sintonización,etc.), el número de canales de video y de audio, en el caso de que el dispositivo pueda capturar imágenes ó el tamaño máximo y mínimo en pixels de la imagen que puede capturar. Para ello utilizamos la llamada `ioctl(fd, VIDIOCGCAP, &cap)`, donde CAP es una variable video_capability.

Además, debemos conocer también las capacidades del canal con el que vamos a trabajar usando la llamada `ioctl(fd, VIDIOCGCHAN, &chan)`. Dicha información queda guardada en video_channel. Para seleccionar el canal adecuado usamos `ioctl(fd, VIDIOCSCHAN, &chan)`.

Podemos obtener las capacidades del sintonizador con los parámetros VIDIOCGTUNER y VIDIOCSTUNER. Aunque para este proyecto, el sintonizador no ha sido utilizado.

Para la selección del modo de funcionamiento disponemos de las siguientes llamadas. Con `ioctl(fd, VIDIOCSWIN, &win)` y `ioctl(fd, VIDIOCGWIN, &win)` declaramos sobre qué región vamos a trabajar. Es decir, si es sobre todos los frames o sólo los pares, definir regiones de pantalla, controlar el efecto clipping, etc.

Y con `ioctl(fd, VIDIOCGPICT, &vpic)` y `ioctl(fd, VIDIOCSPICT, &vpic)` realizamos la consulta y ajuste de brillo, contraste, profundidad de bits y tipo de mapa de bits. Nótese que estos parámetros se suelen diferenciar sólo en la letra G y S. Donde G se refiere a “get” y S se refiere a “Set”.

Tras esto, hemos de elegir la forma en la que vamos a acceder a la imagen que proporciona el dispositivo de captura. Para capturar imágenes podemos selecionar dos modos de trabajo:

- Captura simple: mediante la función `read()`, cuyos inconvenientes son la lentitud y el mayor consumo de recursos del sistema. La función read() nos vuelca un frame a la zona de memoria que le especifiquemos en la llamada. El buffer debe estar ajustado a las condiciones que hemos programado anteriormente. Nótese que no todas las tarjetas soportan dicha función.

- Captura continua: mapeando el vídeo en una región de memoria de nuestro programa, donde el driver volcará los frames de la imagen por DMA desde la tarjeta capturadora y nuestro programa podrá leer, consultar y acceder a esa región de su memoria, leyendo así la imagen. El mapeo de memoria es una funcionalidad que ofrece el kernel de linux. La función mmap intenta ubicar la cantidad indicada de bytes desde el fichero apuntado en la dirección que se le pase como parámetro. Normalmente no se ubican estos bytes exactamente en dicha dirección, por eso `mmap` devuelve la dirección exacta donde ha logrado copiarlos.

En nuestro caso elegimos la segunda opción, reservar la memoria necesaria para que el driver vuelque las imágenes en ella. Para ello utilizamos map = mmap(0,gb_buffers.size,

PROT_READ—PROT_WRITE, MAP_SHARED,fd,0).

Con esto obtenemos una duplicación de la zona de memoria donde el driver almacena la imagen, y copiamos esta duplicación en una región de memoria con la que podemos trabajar.

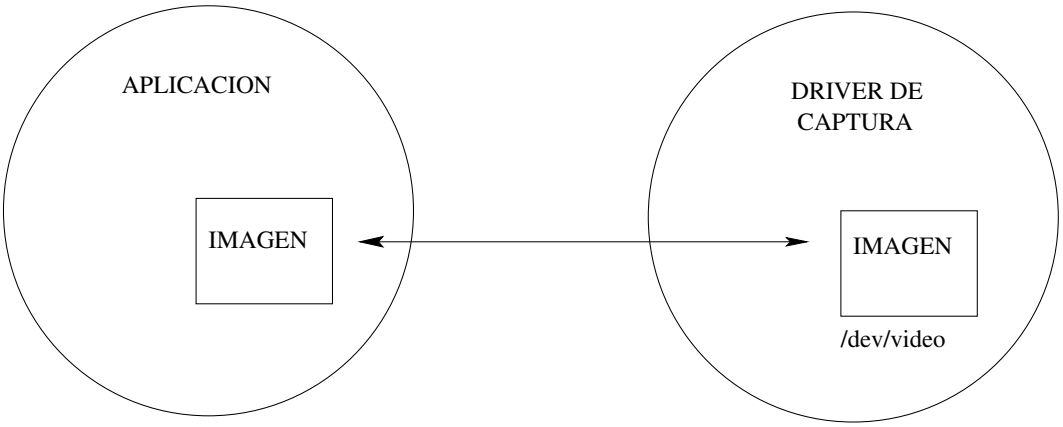

Figura 3.2: Transferencia de la imagen desde el driver a la memoria reservada por la aplicación

Y mediante los parámetros de ioctl VIDIOCSBUF y VIDIOCGBUF inicializamos el buffer de captura. Con VIDEOGCCAPTURE arrancamos o paramos la captura continua de la imagen.

Un ejemplo del código típico de captura sería el siguiente:

```
struct video_mmap gb;
static struct video_mbuf gb_buffers;
...
/* Apertura e inicializacion del dispositivo de video4linux */
fd = open(video, O_RDWR);
if (fd ¡0)
perror(video);
exit(-1);
if (ioctl(fd, VIDIOCGCAP, &cap) ¡0)
perror("VIDIOGCAP");
fprintf(f1, "("video "not a video4linux device?)$n");
close(fd);
exit(1);
for(i=0; i¡cap.channels; i++)
chan.channel=(int) i;
if (ioctl(fd, VIDIOCGCHAN, &chan) == -1)
perror("VIDIOCGCHAN");
close(fd);
```

exit(1);

/* Selecciona el canal composite-video (=1) como canal activo. Si pongo canal Television (=0) tambien funciona. Si pongo s-video (=2) se ve, pero en niveles de gris nada mas */

printf(" %s$n",cap.name);

chan.channel=1;

if (ioctl(fd, VIDIOCSCHAN, &chan) == -1)

perror("VIDIOCSCHAN");

close(fd);

exit(1);

if (ioctl(fd, VIDIOCSWIN, &win) ¡0)

perror("VIDIOCSWIN");

if (ioctl(fd, VIDIOCGWIN, &win) ¡0)

perror("VIDIOCGWIN");

close(fd);

exit(1);

/* Caracteristicas de la imagen, incluyendo formato de pixels!!. Se solicitan al driver y este nos dice que si puede darnoslo o no */

if (ioctl(fd, VIDIOCGPICT, &vpic) < 0)

perror("VIDIOCGPICT");

close(fd);

exit(1);

/*Selección del formato de captura: 24 bits de profundidad*/

if(ioctl(fd, VIDIOCSPICT, &vpic)==-1)

perror("v4l: unable to find a supported capture format");

/* map grab buffer */

if (-1 == ioctl(fd,VIDIOCGMBUF, &gb_buffers))

perror("ioctl VIDIOCGMBUF");

map = mmap(0,gb_buffers.size,PROT_READ—PROT_WRITE,MAP_SHARED,fd,0);

/*Lectura de la imagen en zona de memoria seleccionada en el mapeo*/

src = map+gb_buffers.offsets[gb.frame];

Para salir, procedemos a liberar todos los recursos ocupados y a cerrar los dispositivos.

3.2. Librería Xforms

El sistema X-Windows, también llamado X ó X11, está considerado uno de los más importantes estándares de sistemas de ventanas para Linux y Unix. Su capacidad para independizar a las aplicaciones del hardware de visualización (PC), su transparencia

ante funcionamiento remoto y su soporte para diferentes entornos de trabajo justifica su continuo crecimiento. Además se trata de un sistema de libre distribución y de código abierto.

El diseño del protocolo X especifica una relación cliente-servidor entre una aplicación y su display. El servidor X controlaría, en una máquina, aquello que se muestra por pantalla, así como la localización del ratón y la escritura en pantalla a través del teclado. El cliente X sería cualquier aplicación que envíe peticiones al servidor X, por ejemplo, para mostrar algo en la pantalla o solicitar algo al usuario humano. El servidor acepta peticiones de múltiples clientes y responde a dichas peticiones, comunica las acciones del usuario o envía posibles mensajes de error a las aplicaciones.

Por tanto, el servidor X trabaja en la máquina local, acepta y responde peticiones de clientes, tanto locales como remotos, muestra en pantalla aquello que soliciten los clientes, maneja las acciones del usuario a través del ratón y del teclado (llamadas eventos) y por último, crea y destruye ventanas a petición de los programas clientes.

El cliente consiste básicamente en la aplicación que, basándose en ciertas librerías, utiliza los servicios del servidor.

X-Windows es un sistema muy versátil pero a la vez muy complejo. Esta complejidad no es práctica a la hora de trabajar en aplicaciones comunes, por lo que nos interesa algún tipo de abstracción que haga de interfaz entre el programador y X-Windows, de forma que podamos alcanzar nuestos objetivos con mayor eficiencia. Para ello usaremos la librería Xforms.

Xforms

Desarrollar interfaces de usuario se ha convertido para los desarrolladores de software en algo imprescindible y, debido a la complejidad de los entornos de ventanas, costoso. Por ello, en los últimos años se han desarrollado multitud de paquetes de ayuda en la construcción de interfaces gráficos. Xforms destaca entre aquellos creados para trabajar con X-Windows por ser intuitivo, fácil de usar, potente, gráficamente atractivo y fácilmente ampliable. Es posible crear interfaces gráficos trabajando directamente sobre X-Windows y su librería básica X-Lib, pero dada su complejidad es más ventajoso utilizar esta librería intermedia entre nuestra aplicación y X-Windows.

La librería Xforms ofrece:

a) Un conjunto de objetos para conformar la interfaz gráfica. Estos objetos tienen características visuales y de funcionamiento que pueden ser controladas mediante ciertas funciones que también ofrece Xforms, para controlar su estado o su configuración desde programa, así como para crearlos.

b) Además, para crear estos objetos, Xforms ofrece un diseñador gráfico de interfaces, que utiliza las funciones de la librería.

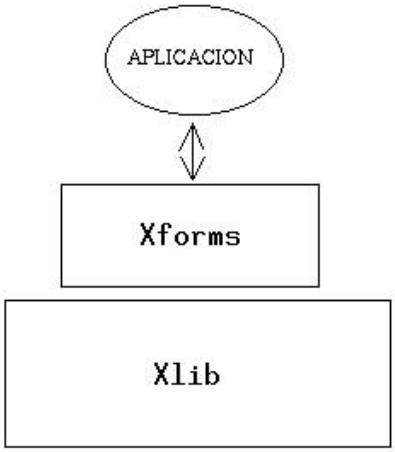

Figura 3.3: Situación de Xforms dentro del sistema

Xforms nos ofrece diferentes elementos gráficos que permiten crear interfaces sobre X-Windows de forma sencilla. Mediante elementos como botones, sliders, canvas, diales y otros objetos tanto de actuación sobre el interfaz como de visualización de gráficos, texto, etc., podemos construir visualmente el entorno gráfico de nuestra aplicación con una herramienta que propociona Xforms llamada Fdesign. Podemos observar un ejemplo de estos objetos en la figura

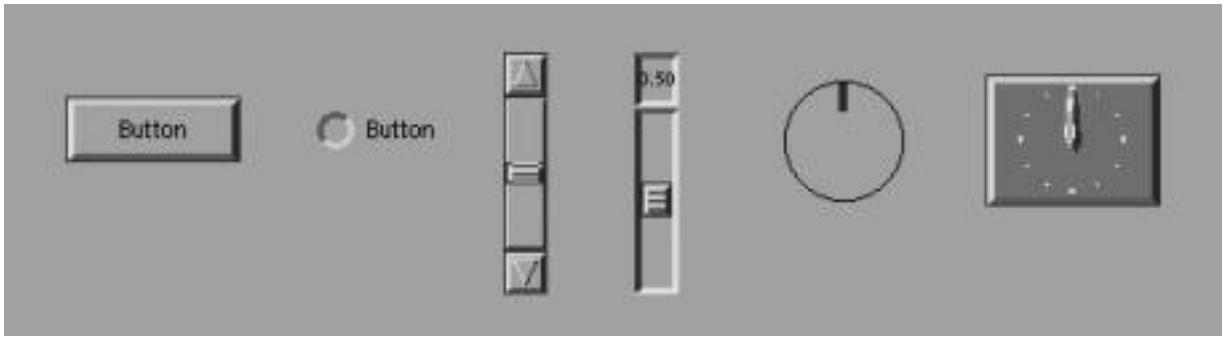

Figura 3.4: Objetos de Xforms

Junto con dichos objetos, Xforms tiene implementadas rutinas que rigen su comportamiento en caso de interacción con el usuario a través del teclado, el ratón o con la propia aplicación para la cual se haya diseñado el interfaz. Estas rutinas facilitan la comunicación entre el interfaz, y por lo tanto el usuario, y la aplicación que se esté creando.

Un elemento muy común de los interfaces es el botón gráfico. Si insertamos un botón en nuestro interfaz con Fdesign, se creará la siguiente llamada en el fichero correspondiente: FL_OBJECT *fl_add_button(int type, FL_Coord x, FL_Coord y, FL_Coord w, FL_Coord h, const char *label). Existen funciones similares para cada tipo de botón.

Para cambar el estado del botón llamaremos a la función void fl_set_button(FL_OBJECT

*obj, int pushed), y para ver en qué estado se encuentra en determinado momento usaremos la función int fl_get_button(FL_OBJECT *obj).

El `slider` también es un elemento gráfico bastante utilizado. El slider sencillo se crea con la función FL_OBJECT *fl_add_slider(int type, FL_Coord x, FL_Coord y, FL_Coord w, FL_Coord h, const char *label). Al igual que en el caso del botón, existen varios tipos de slider y pueden ser manejados con las funciones: void fl_set_slider_value (FL_OBJECT *obj, double val) con la cual ajustamos el slider al valor deseado. O también double fl_get_slider_value(FL_OBJECT *obj) con la que obtenemos el valor que marca el slider en el momento de la llamada.

Con Fdesign disponemos de todo tipo de objetos o elementos gráficos que podemos incluir en el interfaz. Incluso es posible diseñar nuevos elementos. Su funcionamiento consiste en posicionar sobre una ventana de trabajo aquellos objetos que nos interese incorporar al interfaz, modificando con el ratón tanto su posición como su tamaño. También podemos cambiar algunas características de los objetos como tipo y tamaño de letra, color, etc. Su propio interfaz se muestra en la figura 3.5.

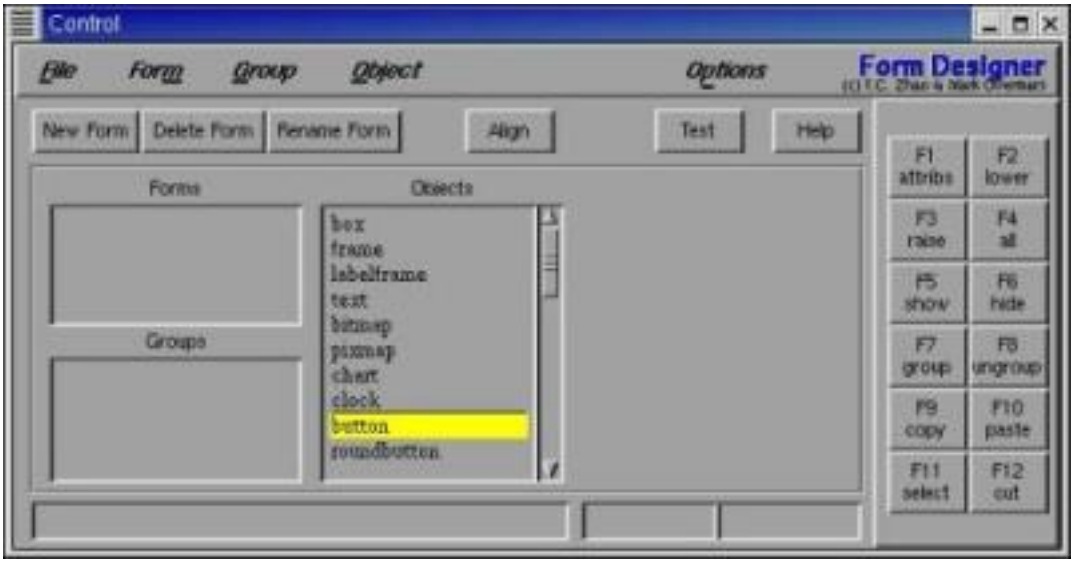

Figura 3.5: Interfaz Fdesign

Tras elaborar una interfaz, el propio Fdesign crea los ficheros .c y .h que materializan el interfaz concreto diseñado y junto con la librería Xforms, formarán parte de nuestra aplicación.

En los entornos de ventanas el usuario tiene mayor facilidad para interactuar con el sistema, ya que además del teclado, tiene el ratón para manejar la aplicación. Por tanto, tanto las pulsaciones de teclas como los movimientos del ratón se engolban en un concepto denominado evento. Cualquier evento debe ser respondido por el sistema o la aplicación de forma rápida. Esto impone una nueva forma de programación. Aparte de la clásica programación secuencial, encontramos una programación basada en eventos. En ella el programa reacciona a los actos del usuario sobre el interfaz ejecutando rutinas

que marcan el funcionamiento de la aplicación. Es decir, no hay flujo predeterminado de control.

Xforms permite trabajar en ambos modos de programación. O bien hacer que la aplicación responda únicamente a los eventos que genere el usuario, ejecutando las rutinas, llamadas `callbacks`, implementadas para dichos eventos (función `fl_do_forms()`, que es bloqueante y entrega el flujo al modo reactivo). O bien mantener un flujo de ejecución lineal y comprobando cada cierto tiempo si se ha producido algún evento mediante la función `fl_check_forms()`.

En el caso de este proyecto, y dada la sencillez del interfaz y que el programa ha de hacer más cosas (leer imágenes, ejecutar el filtro...), hemos optado por la solución no bloqueante. El programa mantiene su flujo de trabajo y periodicamente muestrea los eventos que se han producido desde la última iteración y continúa dicho flujo una vez tratada la petición el usuario, ya que el configurador debe realizar muchas acciones (captura de imágenes, filtrado, etc.) aparte de muestrear y refrescar su propia interfaz gráfica.

Capítulo 4

Implementación

Una vez descritos los objetivos de este proyecto y las herramientas en las que nos hemos basado para su desarrollo, pasamos a explicar el funcionamiento de los diferentes programas y librerías que componen este proyecto. Estos son: una librería de filtrado de color, una aplicación para la calibración de los parámetros de dicho filtro, una librería de segmentación y seguimiento de objetos y por último, un cliente típico de ambas librerías.

Recordemos que este proyecto ha sido desarrollado con la idea de obtener una serie de herramientas de filtrado por color que sean útiles en determinadas situaciones. Por ejemplo, el Grupo de Robótica de la URJC, cuya intención es participar en la competición RoboCup, en la que los robots se identifican principalmente por su color y se verá en la necesidad de llevar su equipo a diferentes instalaciones. En ellas habrá condiciones de iluminación que pueden ser muy diferentes a las que se producen en el laboratorio de robótica del grupo, donde se realizan los ajustes del sistema. Incluso se puede pasar de trabajar con luz artificial a luz natural. Por ello, es necesario poder variar la configuración del filtro de color que estemos utilizando en cada lugar donde se realicen las competiciones, exibiciones, etc.

La librería para filtros de color configurable por fichero consiste en un conjunto de funciones que nos permiten desarrollar aplicaciones de filtrado de colores, muy útiles en robótica. Es un filtro configurable, y sus parámetros se guardan en otro fichero. Cualquier aplicación que realicemos variará los parámetros de trabajo del filtro de este modo. Trabaja sobre tres diferentes espacios de color (RGB, HSI y Lab) simultaneamente, como ya explicaremos más adelante.

La herramienta de configuración del filtro y visualización de resultados utiliza la librería anterior y ofrece un interfaz gráfico que permite realizar un ajuste del filtro de manera sencilla, visualizando los resultados en tiempo real. Para ello, esta herramienta se encarga también de capturar la imagen de la cámara, a la cual podremos aplicar el filtro de color, mostrando al usuario tanto la imagen capturada como la filtrada. Este será el sistema que utilizaremos para configurar los parámetros del filtro antes de poner a funcionar el equipo para la RoboCup.

La librería de segmentación y seguimiento de objetos está compuesta de funciones relacionadas con el agrupamiento de pixels del mismo color en segmentos rectangulares y predicción de trayectorias. Están pensadas para ser utilizadas sobre una imagen a la cual se le han aplicado anteriormente las funciones de la librería del filtro de color, ya que utiliza variables en las que se indica el número de filtros que están activos y el color con el que están representadas las etiquetas.

El cliente típico utiliza las dos librerías anteriores con el fin de tratar la imagen proveniente de la cámara mediante un filtro de color. Tras este proceso el cliente aplica las funciones de segmentación y seguimiento de la librería anterior. Sirve de esqueleto para facilitar a los usuarios de las dos librerías anteriores un ejemplo concreto de su uso. Así, la creación de un nuevo cliente más completo se acelera al contar con este código ya escrito

En este capítulo describiremos en detalle estos cuatro elementos que hemos desarrollado dentro de este proyecto.

4.1. Filtros de color

El filtro de color es útil dentro del campo de la robótica ya que el color es una primitiva muy util para extraer información relevante del entorno. Muchos robots tienen como sensor principal un dispositivo de visión. El Grupo de Robótica de la URJC trabaja con algunos como el EyeBot y el Aibo. Con el robot EyeBot hemos desarrollado varios proyectos que consistían principalmente en filtros de color para éste como el proyecto Sigue Pared de Victor Gómez [Gómez02] , en el que se filtraba el color de la pared para hacer al robot que la siguiera, salvando esquinas. O también el proyecto Sigue Pelota, de Félix San Martín [SanMartín02], que permitía al robot EyeBot seguir una pelota naranja filtrando su color.

El color se define desde las capacidades del ojo humano. Es decir, si se considera a un ser humano normal, su visión de un color será la misma que la de otro ser humano normal. Pero para dar información sobre un color se necesita una definición o un modelo. Desde un punto de vista abstracto, una imagen de color puede ser considerada como una función F, donde F(x,y) proporciona el color del pixel en la posición (x,y). Para imágenes en blanco y negro, F(x,y) será un valor simple, que representa la escala de grises del punto. Para imágenes en color, F(x,y) será una tupla que representa un color perteneciente a un modelo de color determinado. Un modelo de color es un método para especificar colores con respecto a una estructura de referencia. Existen varios tipos de modelos de color. En este proyecto utilizaremos los siguientes: modelo RGB, modelo HSI y modelo Lab, cuyas características describiremos más adelante.

El modelo RGB es apropiado para su uso en monitores, pero para el ser humano

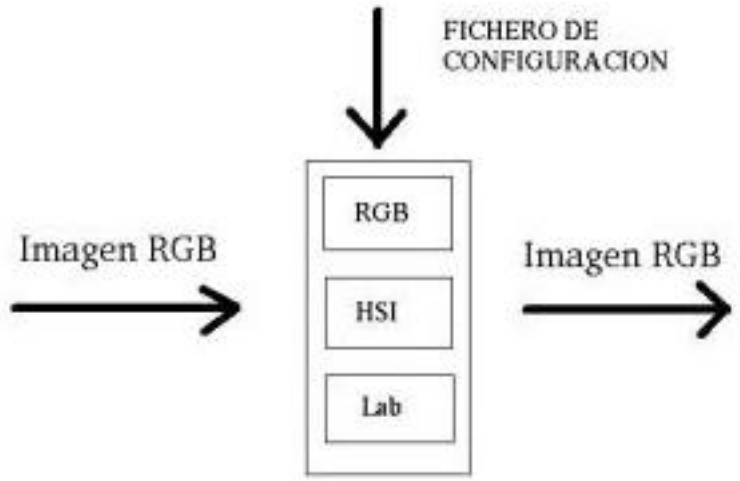

Figura 4.1: Esquema de librería para filtros de color

es más útil el espacio HSI para la manipulación de imágenes en color. Además hay otros espacios de color, como el modelo CMY, que son útiles, por ejemplo, para el funcionamiento de las impresoras. Por este motivo, las conversiones entre modelos son importantes. La conversión entre espacios de color se puede interpretar como una transformación de vectores, gracias a las interpretaciones geométricas de los modelos de color y en teoría se pueden realizar transformaciones entre cualquier espacio de color, aunque en algunos casos son más complejas que en otras.

Estructura de la librería

Como hemos visto, podemos modelizar el color mediante cierto número de componentes, dependiendo del espacio de color en el que nos basemos (ver figura 4.1). La interpretación de dichos valores dependerá del modelo de color utilizado. En estos tres modelos tratados (RGB, HSI, LAB), se puede interpretar un color como un vector tridimensional. En los tres modelos anteriores se puede realizar la selección de una región del espacio de color acotando el valor de los tres parámetros a unos rangos determinados. Es decir, para un cierto color en cualquier espacio, definido por tres valores, podemos seleccionar tres rangos, uno para cada parámetro, dentro de los cuales estará incluído este color. Los filtros implementados no definen zonas arbitrarias en el espacio de color. Al tratar cada eje de manera independiente a los otros dos, tenemos filtros definidos como paralelepípedos en el espacio RGB, sectores truncados en el HSI, etc., donde todo lo que caiga en esa zona se considera esencialmente del mismo color.

De esta manera estamos marcando un criterio para poder valorar si un determinado color se asemeja en mayor o menor medida, dependiendo de la amplitud de los rangos, a otro color. Este es el funcionamiento básico de los filtros de color utilizados en este proyecto.

Una imagen se almacena en el computador como una matriz discreta de pixels, donde cada pixel se representa como un valor aceptado en dicho modelo de color. Si aplicamos un filtro de color a cada uno de los pixels de la imagen, seleccionando sólo aquellos cuyo color se asemeje a un color previamente definido, podremos eliminar de la imagen aquellos colores que no se parezcan, según el criterio anterior, al color elegido. El hecho de eliminar un color de la imagen en realidad no se lleva a cabo, pero se pueden reescribir aquellos pixels que no superen el filtro por ejemplo en color negro. Como podemos estar filtrando en una imagen varios colores a la vez, es necesario especificar previamente un color representante para cada uno de ellos.

La salida del filtro es otra imagen donde cada pixel ha sido evaluado: para colores parecidos al color que queremos filtrar, se sustituye el pixel por el color representante del color a filtrar. Para colores distintos a éste, se cambia el color del pixel por el negro. De este modo, tras realizar esto mismo con todos los pixels de la imagen, obtendremos otra imagen, con el mismo número de pixels, en la que se han convertido en negro todos los pixels excepto aquellos que hayan superado el filtro. Es decir, aquellos cuyos tres valores característicos entran dentro de los rangos definidos alrededor de los valores característicos del color que queremos filtrar, que aparecen con los valores del color representante.

Por tanto, el proceso de filtrar una imagen en cualquiera de los tres espacios de color elegidos es similar.

1. Elección del color a extraer de la imagen y obtención de sus valores característicos respecto del modelo de color a utilizar. Esto depende de la aplicación concreta que se le vaya a dar. Por ejemplo, la pelota naranja para la RoboCup o colores chillones como el fucsia y el cian para las balizas de localización.

2. Elección de unos rangos alrededor de dichos valores. Es importante que estos rangos no sean ni demasiado amplios, ya que entonces superarían el filtro colores que no se asemejan realmente al color elegido, ni demasiado estrecho, pues sólo lo superarían los colores idénticos al elegido. También es necesario seleccionar un color como representante del rango de colores que vamos a filtrar.

3. Comprobación con cada pixel de la imagen, de si sus tres parámetros se encuentran dentro del rango definido anteriormente. En caso afirmativo, ese pixel superará el filtro y no será eliminado de la imagen.

En el caso de que el filtro utilizado sea diferente del RGB, que es el que toma la biblioteca como espacio de trabajo, habrá que realizar una transformación de los datos de la imagen al espacio de color que se trate, antes de realizar esta comprobación. El espacio RGB ha sido elegido como espacio de trabajo debido a la coincidencia con el espacio en el que se realiza tanto la captura como la visualización. Puesto que en este caso, el modelo de color en el que se realiza la captura coincide con el del filtro, no

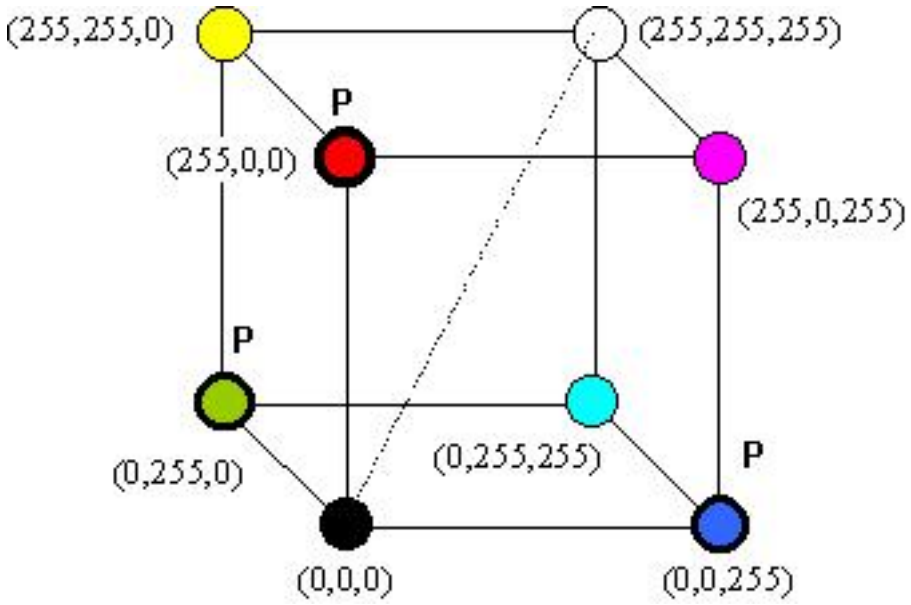

Figura 4.2: Representación del espacio RGB

será necesario operar sobre la información almacenada en la imagen para realizar el filtrado.

El filtro de color, por tanto, debe comprobar si cada elemento de dicha matriz cumple las condiciones necesarias para superar el filtro. Estas condiciones difieren según sea el tipo de filtro utilizado. Puesto que en esta aplicación nos interesará filtrar varios colores y utilizar varios tipos de filtro simultaneamente aprovecharemos un sólo barrido de la imágen para realizar todas estas operaciones.

4.1.1. Filtro RGB

El modelo de color RGB es dependiente del dispositivo y suele ser usado por los monitores de televisión y computadoras. Tiene tres colores primarios básicos: rojo (R), verde (G) y azul (B). Todos los demás colores son obtenidos mediante la combinación de estos tres colores (modelo aditivo). Un color particular se define en RGB como la "cantidad" de estos tres colores primarios que lo componen, por ejemplo, el color amarillo se puede obtener a partir de la mezcla de rojo y verde. La descripción de este modelo mediante tres parámetros posibilita su representación mediante un espacio geométrico tridimensional. En este caso todos los colores se representan dentro de un cubo euclídeo, donde las tres esquinas perpendiculares y no adyacentes son R,G y B, que son los colores puros ó básicos de este modelo (ver figura 4.2). Esta representación geométrica facilita su interpretación y más adelante se comentará la aplicación que se ha dado a esta propiedad en la configuración del filtro de color.

Simplemente tomaremos, para cada pixel, cada uno de los tres parámetros (Red, Green y Blue) y comprobaremos si pertenecen a los rangos definidos para el color a filtrar. Lógicamente, al tener la posibilidad de definir diferentes etiquetas (diferentes

colores a filtrar), comprobaremos la pertenencia a los rangos de todas estas etiquetas que tengamos activas. El código para implementar el filtrado de cada color es aproximadamente:

if ((RGBData[3*k+2]>rangosRGB[n][0])&&(RGBData[3*k+2] <=rangosRGB[n][1])
&&(RGBData[3*k+1]>rangosRGB[n][2])&&(RGBData[3*k+1] <=rangosRGB[n][3])
&&(RGBData[3*k]>rangosRGB[n][4])&&(RGBData[3*k]<=rangosRGB[n][5]))

Imagen_filtrada[k*3+2]=samplesRGB[n][0];

Imagen_filtrada[k*3+1]=samplesRGB[n][1];

Imagen_filtrada[k*3]=samplesRGB[n][2];

Donde RGBData es el array bidimensional que almacena los datos de la imágen; k es el número del pixel que estamos analizando en este barrido; n es el número de la etiqueta que estamos comprobando; Imagen_filtrada almacena la imagen una vez filtrada y samplesRGB contiene el color de muestra por el que van a ser sustituídos los pixels que superen el filtro, quedando el resto en negro.

4.1.2. Filtro HSI

El modelo de color HSI fue diseñado teniendo en mente el modo en que artistas y diseñadores gráficos usan los términos "saturación" (la pureza del color), "tono" (el color en si mismo) e "intensidad" (el brillo del color). Esto es exactamente lo que el modelo de color HSI representa. Geométricamente, se interpreta como un doble cono.

En HSI, las componentes HS son relativamente invariables frente a cambios de iluminación de la escena, que quedan absorbidos fundamentalmente por la componente I. Esto favorece la robustez de los filtros HS ante esas variaciones.

La principal desventaja de este modelo, desde el punto de vista computacional, es que no suele estar en el formato nativo en el que entregan los datos las tarjetas digitalizadoras de video, por lo tanto es necesaria una conversión, lo cual requiere cierto tiempo de cómputo. En nuestro caso, nos interesa poder realizar conversiones hacia diferentes espacios de color que presenten una mayor invarianza respecto a cambios de iluminación.

En este caso, el modelo de color de la captura no coincide con el del filtro, por lo que tenemos que hacer una transformación con los datos de la imagen, obteniendo mediante fórmulas matemáticas, el tono y la saturación equivalentes a los valores RGB almacenados en la imagen. Las siguientes igualdades resuelven dicha transformación.

Las siguientes fórmulas indican el método de con versión entre los espacios de color HSI y RGB.

Conversión de RGB a HSI:

I = 1/3(R+G+B)

S = 1 - (3/(R+G+B))*a donde a es el mínimo entre R, G y B

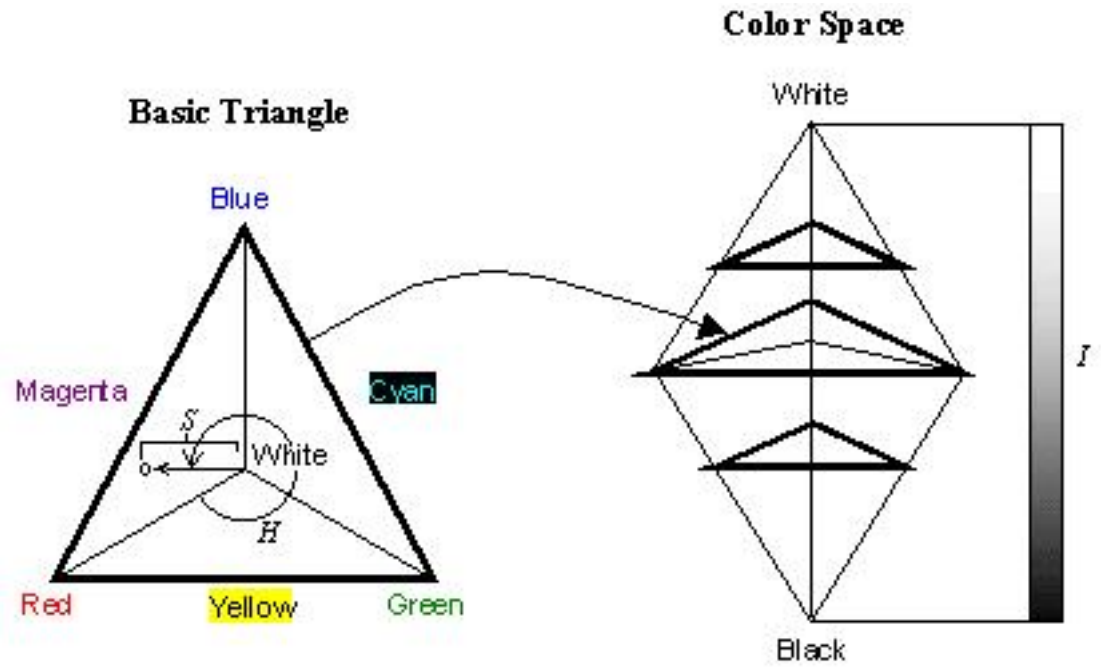

Figura 4.3: Representación del modelo de color HSI

H = cos$\hat{}$(-1) ((0.5*((R-G)+(R-B))) / ((R-G)$\hat{}$2 + (R-B)*(G-B))$\hat{}$(0.5))

Si S = 0, H carece de significado.

Si (B/I) > (G/I) entonces H = 360 - H puesto que H es un ángulo en grados normalizamos a 0,1 con H=H/360

Conversión de HSI a RGB:

Primero restauramos H a grados con H = 360*H

If 0 < H <= 120 then

B = 1/3(1-S) R = 1/3(1+ [(S cos H) / (cos(60 - H))]) G = 1 - (B+R)

If 120 < H <= 240 then

H = H - 120 R = 1/3(1-S) G = 1/3(1+ [(S cos H) / (cos(60 - H))]) B = 1 - (R+G)

If 240 < H <= 360 then

H = H - 240 G = 1/3(1-S) B = 1/3(1+ [(S cos H) / (cos(60 - H))]) R = 1 - (G+B)

El parámetro I (intensidad) no lo calculamos ya que en este caso sólo realizaremos el filtrado de el tono (H) y la saturación (S). Una vez obtenidos estos valores llevaremos a cabo el filtrado del mismo modo que en el apartado anterior. El código simplificado quedaria:

```
r=(double)RGBData[3*k+2]/255.;
g=(double)RGBData[3*k+1]/255.;
b=(double)RGBData[3*k]/255.;
rgb2hs(r,g,b,&H,&S);
if ((H>=rangosHSI[m][0]) && (H <=rangosHSI[m][1])
```

&& (S>=(rangosHSI[m][2]/128.)) && (S<=(rangosHSI[m][3]/128.)))

Imagen_filtrada[k*3+2]=samplesHSI[m][0];

Imagen_filtrada[k*3+1]=samplesHSI[m][1];

Imagen_filtrada[k*3]=samplesHSI[m][2];

En r,g y b almacenamos el valor de los parámetro del pixel tratado y dentro del procedimiento rgb2hs aplicamos las fórmulas anteriores para obtener H y S.

Además, se incluye para hacer la librería más general. En el modelo RGB, un mismo color bajo diferentes condiciones de iluminaciones presenta grandes diferencias en sus tres componentes, mientras que el modelo HSI mantendrá relativamente estables las componentes de tono y saturación.

En algunos casos, como ya veremos más adelante, nos interesa dar cierto margen para poder capturar colores que, pese a ser iguales (pertenecen al mismo objeto de color uniforme), reciben una iluminación diferente, lo que hace variar sus parámetros y por lo tanto, no superar el filtro.

4.1.3. Filtro Lab

Lab es un modelo que describe el color en tres planos: L, a y b. Este modelo no está orientado hacia ningún dispositivo de salida como los mencionados anteriormente. Por tanto está considerado como un modelo independiente del dispositivo.

- L: describe la claridad del color (rango de 0 a 100; números más altos implican mayor claridad).

- a: describe la cantidad de rojo o verde presente en el color (rango entre -128 y 127; los valores positivos son más rojos y los negativos son más verdes).

- b: describe la cantidad de amarillo o azul presente en el color (rango entre -128 y 127; los valores positivos son más amarillos y los negativos son más azules).

Para implementar este filtro de nuevo es necesario hacer una conversión del espacio de color RGB al Lab. Nos interesan los parámetros a y b, ya que, sólo filtraremos estos dos valores. Las siguientes fórmulas nos permiten calcularlos.

La conversión se realiza en dos fases. Primero se convierte de RGB a XYZ mediante:

[X] [0.412453 0.357580 0.189423] [R] [Y] = [0.212671 0.715160 0.072169] * [G] [Z] [0.019334 0.119193 0.950227] [B]

Después se pasa de XYZ a Lab:

L* = 116 * (Y/Yn)$\hat{}$(1/3) - 16 for Y/Yn ¿0.008856 L* = 903.3 * Y/Yn en otro caso.

a* = 500 * (f(X/Xn) - f(Y/Yn)) b* = 200 * (f(Y/Yn) - f(Z/Zn)) donde f(t) = t$\hat{}$(1/3) para t ¿0.008856 f(t) = 7.787 * t + 16/116 en otro caso.

El código para implementar este filtro es similar al siguiente:

RGB2Lab(RGBData[3*k+2],RGBData[3*k+1],RGBData[3*k],&L,&A,&B);

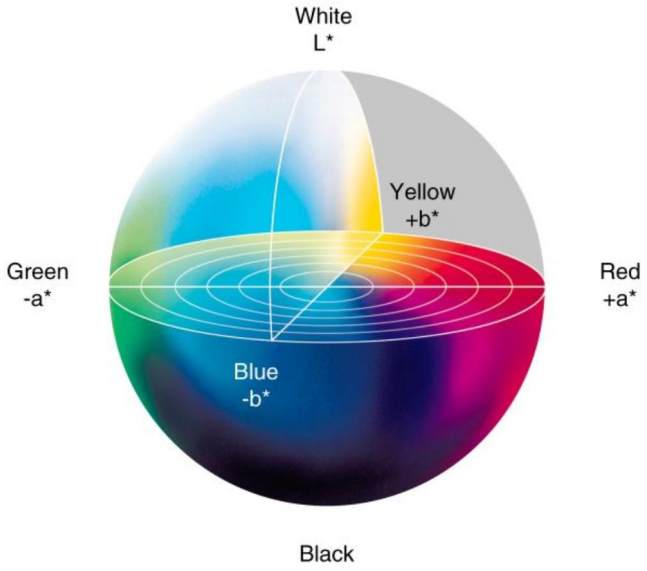

Figura 4.4: Representación del modelo de color Lab

```
if ((A>rangosLab[s][2])&&(A<=rangosLab[s][3])
&&(B>rangosLab[s][4])&&(B<=rangosLab[s][5]))
    Imagen_filtrada[k*3+2]=samplesLab[s][0];
    Imagen_filtrada[k*3+1]=samplesLab[s][1];
    Imagen_filtrada[k*3]=samplesLab[s][2];
```

Fichero de configuración

Este fichero, llamado `siguecolor.conf` nos permite configurar todos los parámetros del filtro de color implementado en la librería, así como los colores a filtrar, los umbrales, los colores representantes, etc. Como ya se ha explicado antes, para filtrar un color dentro de un espacio de color determinado necesitamos fijar unos rangos para alguna de sus componentes, mediante los cuales, realizaremos el filtrado de los pixels que no estén dentro de los valores especificados. Por tanto, nos interesa poder realizar simultaneamente el filtrado de una imagen en diferentes espacios de color. Esto lo especificaremos en el fichero de configuración, donde almacenaremos con un formato determinado tantos colores como queramos filtrar, indicando el valor de los rangos y el tipo de modelo de color en el que queramos trabajar. Para distinguir en la imagen filtrada aquellos pixels que procedan de cada filtro, especificaremos también un color de muestra para cada color a filtrar.

Este fichero se puede escribir con un editor de textos normal. Para facilitar una selección más intuitiva de los parámetros del filtro hemos desarrollado una herramienta que describiremos en la sección 4.2. Los datos de configuración serán rellenados automáticamente por la herramienta de calibración del filtro, pero es conveniente saber en qué formato funciona ya que en alguna ocasión nos puede hacer falta incluír datos a mano.

El siguiente ejemplo ilustra la forma en que se utiliza el fichero:

#etiquetas para el filtro HSI:
hsifilter verde -1.02 -0.42 81 113 0 255 0
hsifilter amarillo -0.77 -0.17 31 51 255 255 0
#etiquetas para el filtro RGB:
rgbfilter azul 212 245 118 138 0 36 0 171 255
rgbfilter blanco 178 198 90 127 50 102 255 255 255
#etiquetas para el filtro Lab:
labfilter rojo 58 78 3 23 58 78 255 0 0
#variable para el configurador:
threshold 13
#selección del origen de entrada de la imagen:
input v4l

Para cada color que se quiera filtrar se añade una línea de la siguiente forma:
<tipo de filtro> <rango 1>...<rango n> <color representante>

Como tipos de filtro podemos elegir entre: `hsifilter, rgbfilter y labfilter`. Los rangos dependerán de cada filtro. En el caso del HSI tenemos un rango para el tono (H), desde -3.14 hasta 3.14, y otro para la saturación (S), de 0 a 127. Para el filtro RGB hay que elegir tres rangos (R, G y B), cada uno entre 0 y 255. Por último, el filtro Lab utiliza también tres rangos, uno para cada parámetro L, a y b, donde L está entre 0 y 100, a y b entre -128 y 127.

En el fichero de configuración se incluyen, además de estas lineas, algunos parámetros que utiliza la herramienta de configuración que veremos más adelante. Por ejemplo, hace falta especificar el número mínimo de pixels de un determinado color que tienen que aparecer en la imagen para que éste color se vea reflejado en el histograma HSI. Esto se indica con la siguiente linea: `threshold <nº de pixels>`.

Por último, hace falta añadir una linea que indique de dónde se debe tomar la fuente de la imagen: con la linea: `input v4l`, indicamos que la fuente será la cámara, mientras que con `input imagefile <path y nombre de fichero>`, se trabajará sobre la imagen del fichero indicado, que debe estar en el formato PPM crudo.

Otro tipo de linea aceptado dentro del fichero de configuración son aquellas empezadas con le símbolo #, que son utilizadas para comentarios.

4.2. Herramienta de configuración

Como ya hemos explicado antes, la herramienta de configuración permite modificar el fichero de configuración del filtro de color para su posterior uso con la librería del

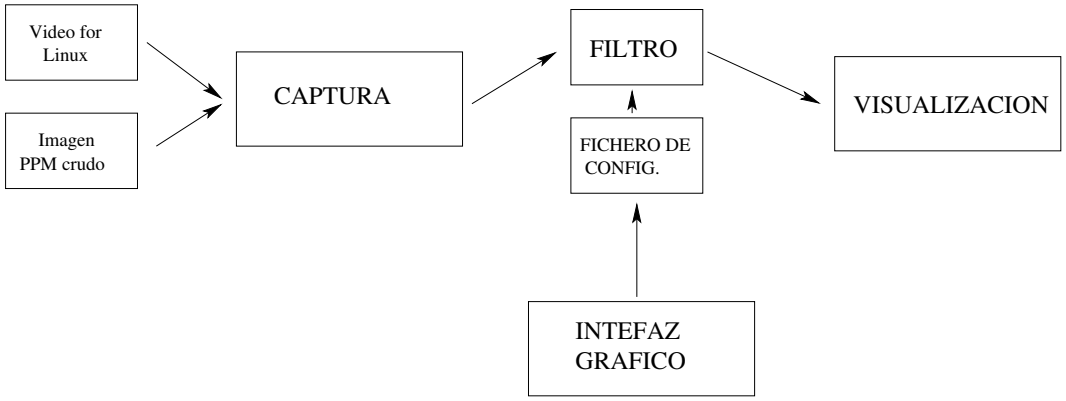

Figura 4.5: Esquema de herramienta de configuracion

filtro. Para ello ofrece una interfaz gráfica sencilla e intuitiva, que además muestra, en tiempo real, el resultado de los cambios realizados. Esta herramienta puede ser útil en diferentes trabajos con visión artificial. Por ejemplo será utilizado en el sistema que está desarrollando el Grupo de Robótica de la URJC para participar en la RoboCup, donde los jugadores se identifican por unas pegatinas de colores. Dentro de este contexto, la herramienta de configuración será utilizada en las competiciones de fútbol robótico que, al ser realizadas en diferentes lugares, estarán sometidas a diferentes condiciones de iluminación como ya se explicó anteriormente. En tales circunstancias, esta aplicación permite ajustar los parámetros del filtro de forma óptima al entorno concreto de la competición. Utilizando la cámara cenital del campo de juego para identificar la pelota, la posición y orientación de los robots, etc.

Esta aplicación ha sido desarrollada utilizando la librería Xforms para generar el interfaz gráfico, la librería se basa en VideoForLinux para la captura de la imagen de la cámara y en la propia librería desarrollada para realizar el filtro (ver figura 4.5).

Al iniciar la aplicación, ésta lee el contenido del fichero de configuración, en el cual puede encontrar información explicada en la figura 4.5

La herramienta de calibración cuenta con tres ventanas:

- Una ventana que muestra la imagen tal como la está capturando la camara ó cómo está en el fichero.

- Otra con la imagen resultante de la aplicación del filtro.

- La ventana principal del configurador, en la que encontramos elementos que nos permiten modificar de forma sencilla todos los parámetros del filtro (4.6).

En esta última ventana están situados los siguientes elementos que describimos a continuación.

Etiquetas

El filtro puede buscar varios colores a la vez, por lo que es necesario definir una etiqueta para cada color. Esta etiqueta consiste en la información relativa a los rangos

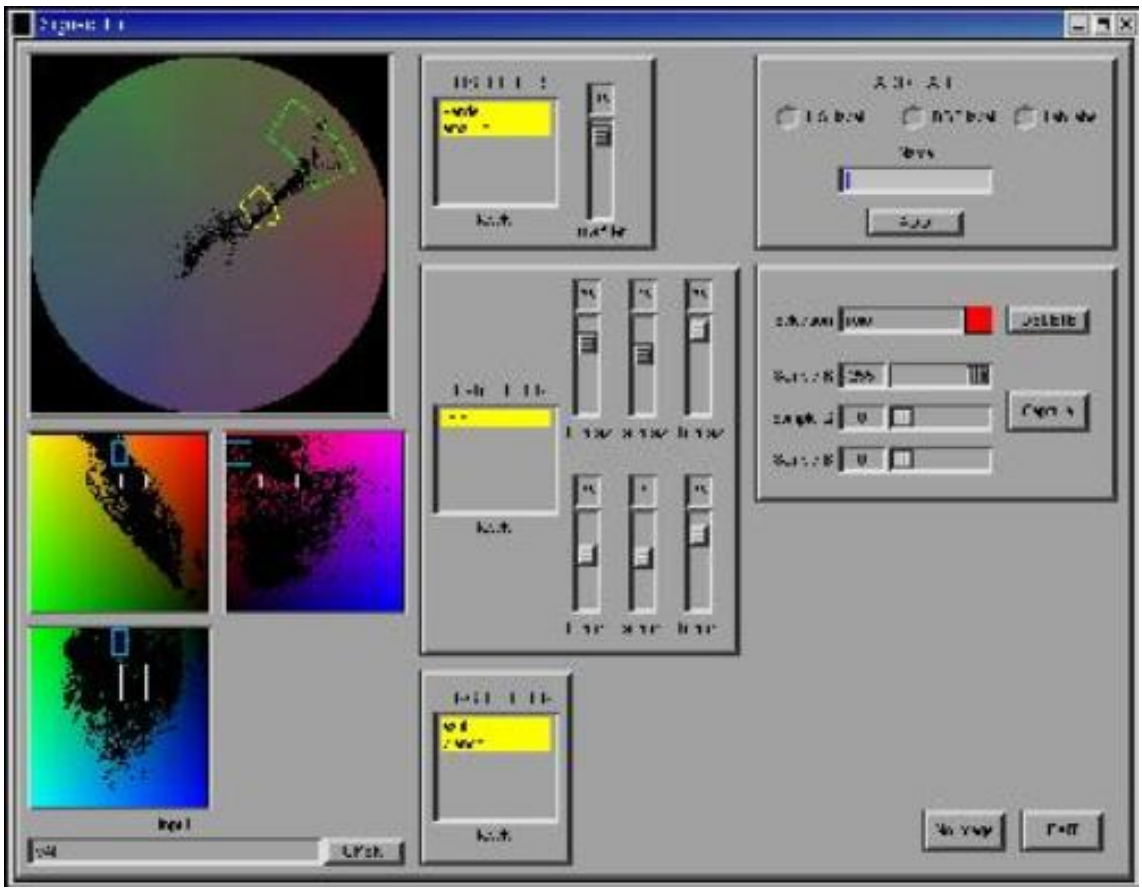

Figura 4.6: Interfaz del configurador del filtro de color

en el espacio al que pertenezca el color a filtrar y el color representante que tendrá dicho color en la imagen de salida. Estos valores se definen en los elementos de la figura 4.11.

Cada etiqueta queda definida por un nombre de etiqueta, que aparece en listado, un rango para el parámetro H y otro para el S y tres valores (R,G y B) correspondientes al color elegido para representar a esa etiqueta tanto en la imagen filtrada como en el histograma HSI. Puesto que realizamos los ajustes de modo visual, los valores numéricos no nos interesan y por tanto, no aparecen escritos en el interfaz (ver figura 4.7). Unicamente indicamos el nombre de la etiqueta para que pueda ser seleccionado y hacer sobre los demás parámetros los cambios necesarios.

Por último, el interfaz nos muestra un apartado en el que podemos ver y modificar el color representante de la última etiqueta seleccionada en las listas de cada filtro, crear nuevas etiquetas y eliminar las existentes de forma rápida. En la figura 4.11 se observan los objetos necesarios para llevar esto a cabo.

Disco de distribución del espacio HSI

Sección circular del espacio de color HSI, en la que se van a situar las areas que representan las diferentes etiquetas o rangos de color que queremos filtrar en la imagen de la cámara. Consiste en un disco en el que se distinguen todos los colores correspon-

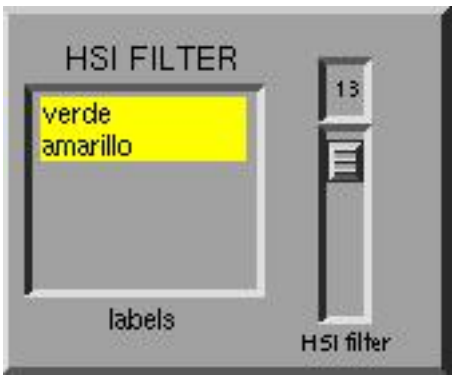

Figura 4.7: Listado de etiquetas HSI

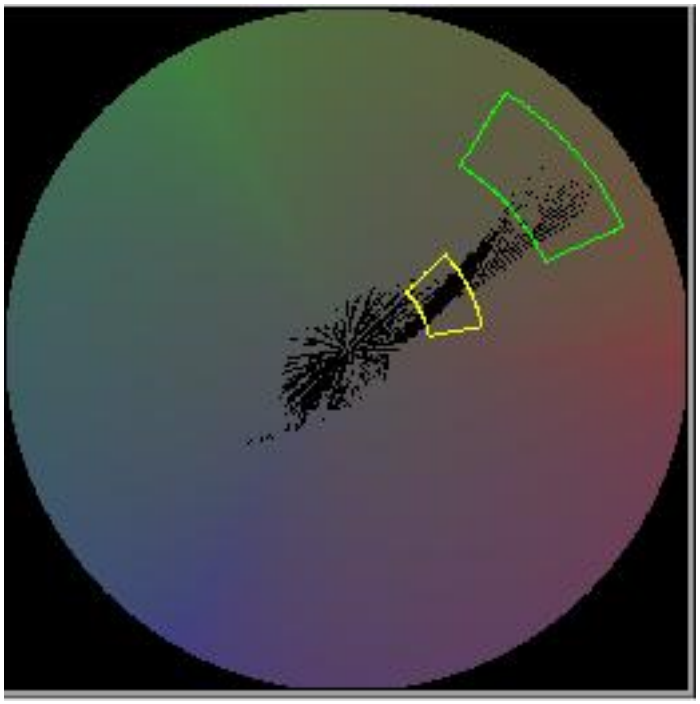

Figura 4.8: Disco de distribución del espacio HSI

dientes a las diferentes combinaciones posibles entre el tono (H) y la saturación (S) y donde la intensidad (I) se ha considerado constante. La saturación varía con el radio del disco y el tono con el ángulo del mismo (ver figura 4.8).

Situaremos en color negro sobre el disco del histograma los valores H y S correspondientes al color del pixel analizado en cada etapa de filtrado, con lo que estaremos visualizando la distribución de los valores H y S que capta la cámara en todo momento. De este modo sabremos por qué zona del espacio HS se sitúa el color de cada objeto, lo que nos dará una grán facilidad a la hora de elegir los parámetros H y S necesarios para configurar el filtro. Esto es útil para poder definir visualmente las etiquetas con sólo observar la distribución del color de la imagen actual.

Así, para ajustar la etiqueta correspondiente, por ejemplo, al “amarillo”, se pone delante de la cámara un objeto amarillo y en el disco de distribución del color en HS se

verá la zona en la que aparece el objeto, ya que los pixels de este color se tornarán en negro siempre que el objeto esté delante de la cámara. De este modo, los parámetros del filtro pueden ajustarse porque cubren esa zona y sólo esa zona, concretando lo que el filtro entiende por amarillo.

Estos rangos se representan también sobre el histograma HSI en forma de segmentos del color representante de la etiqueta, formando una sección poligonal en forma de "quesito". Se ha implementado la posibilidad de cambiar los rangos H y S de cada etiqueta, pinchando con el ratón directamente sobre el disco HSI. Todos aquellos puntos de la imagen cuyos valores equivalentes en H y S caigan dentro de esta sección, estarán superando el filtro de color, con lo que en la imagen filtrada aparecerán dichos puntos, con el color representante de esa etiqueta que hayamos elegido. De este modo, puesto que sabemos visualmente dónde se encuentran los valores H y S del color que queremos filtrar, podemos pinchar con el ratón justo encima de las lineas que muestran los rangos de cada etiqueta y arrastrarlas hasta englobar el area deseada. Esto nos da una gran rapidez en el proceso de localizar los rangos correctos para cada color buscado y ajustar los parámetros del filtro a esos rangos.

Se pudo haber hecho esta configuración numéricamente, pero el configurador debe ser intuitivo. No obstante, se puede definir numéricamente las etiquetas escribiendo en el fichero de configuración del filtro.

Histogramas RG, GB y RB

Estos tres histogramas representan los tres planos que forman el espacio de color RGB, limitados a un tamaño de 128x128 pixels. Se ha elegido esta proyección debido a que mostrar un espacio tridimensional en pantalla es complicadoComo color de fondo tienen toda la distribución de colores posible derivada de la combinación de sus tres elementos, de dos en dos para cada plano. Por ejemplo, en el plano RG encontramos desde el rojo puro hasta el verde puro, pasando por todas las posibles mezclas. Sobre este fondo de color, se resaltan como puntos negros todos aquellos valores que aparezcan en la imagen, ya que un punto RGB tiene proyección en estos planos. Es decir, si un pixel de la imagen tiene como parámetros R=123, G=45 Y B=207, en el plano RG aparecerá en negro aquel pixel correspondiente a las coordenadas R=123 y G=45.

Al igual que en el histograma HSI, podemos localizar visualmente y de forma rápida la localización de un color cualquiera que mostremos delante de la cámara dentro del espacio RGB, con lo que la labor de elegir los rangos para filtrar este color en RGB es inmediata. Y para ello también se ha implementado la funcionalidad de ver en forma de areas delimitadas por lineas sobre los planos RG, GB y RB, los rangos correspondientes a cada color a filtrar, quedando dichos rangos definidos por un rectángulo sobre cada imagen. Así, podemos comparar también visualmente dónde están situados estos rangos

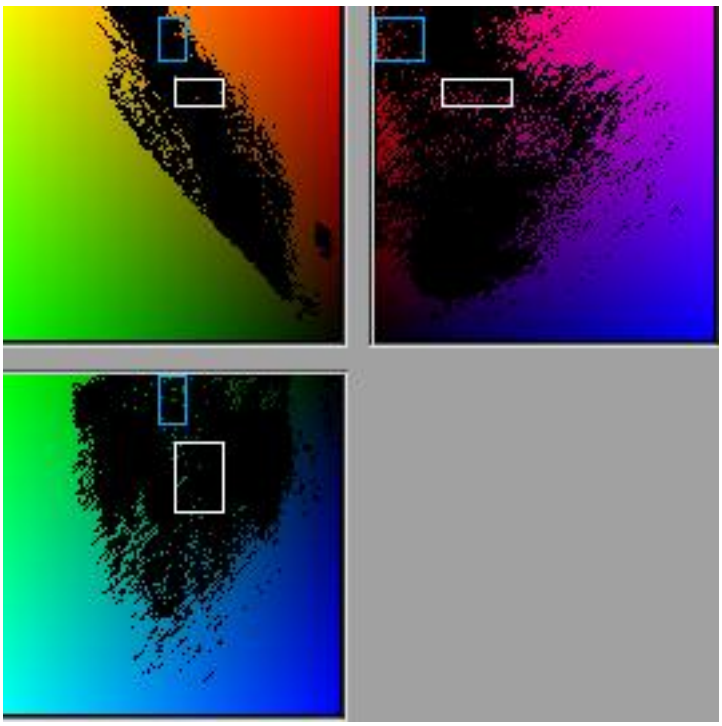

Figura 4.9: Histogramas RGB

dentro del espacio RGB y dónde deberían estar para filtrar el color deseado.

Podemos hacer coincidir ambas cosas simplemente pinchando sobre estas líneas de cada histograma y arrastrándolas al lugar deseado. De nuevo hemos implementado una solución para uno de los principales requisitos de este proyecto: configurar el filtro de color de forma sencilla e intuitiva, ya que no tenemos por qué conocer los valores numéricos de los rangos que estamos eligiendo, sino que simplemente vemos dónde deben situarse dentro del espacio de color, tanto para el HSI como para el RGB.

Junto a estos histogramas también encontramos una pequeña lista donde nos aparecen todas las etiquetas que estamos filtrando en el espacio RGB.

Controles para el filtro Lab

Dada la complejidad de la representación gráfica de este espacio de color, resultaría demasiado costoso en tiempo realizar una interfaz intuitiva similar a las anteriores para este filtro. Por este motivo, se han colocado seis "sliders" que nos permiten modificar numéricamente los rangos correspondientes a cada parámetro del espacio Lab.

Color de muestra

Tal vez, la función más interesante de todo el configurador del filtro sea el botón de captura de color, ya que con sólo pinchar en la ventana que contiene la imagen real, podemos seleccionar directamente qué color queremos filtrar y la propia aplicación definirá unos rangos tentativos alrededor de dicho color. Primero debemos seleccionar alguna de las etiquetas de los filtros y a continuación pulsar el botón "CAPTURE" del

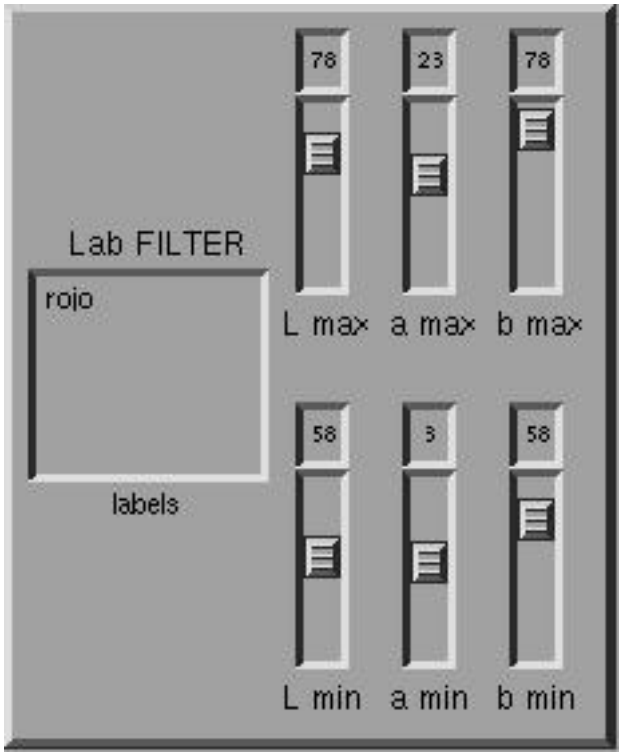

Figura 4.10: Controles para el filtro Lab

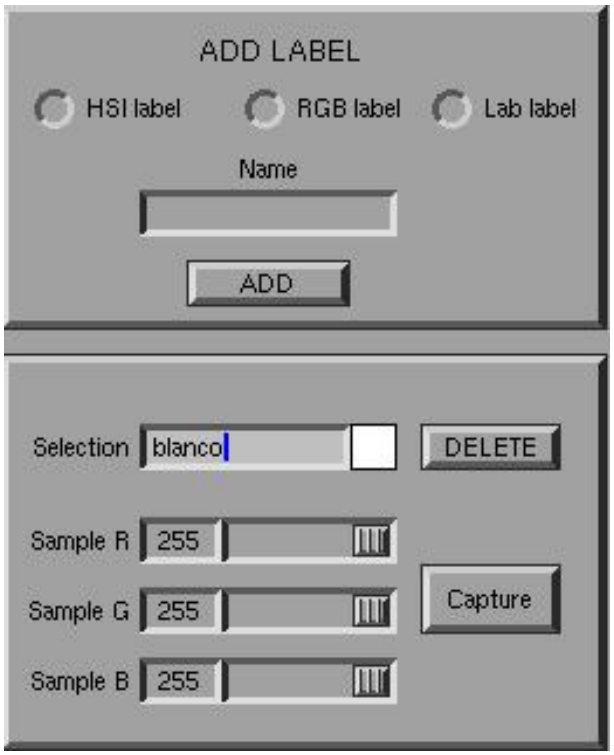

Figura 4.11: Manejo de etiquetas

interfaz. La imagen original de la cámara quedará congelada hasta que pinchemos con el ratón en algún punto de la misma. Lo que ocurrirá entonces es que la aplicación leerá las componentes RGB del pixel que hayamos seleccionado en la imagen. Dependiendo de a qué tipo de filtro pertenezca la etiqueta previamente activada, hará las transformaciones necesarias para calcular, si por ejemplo se trata del filtro HSI, la H y la S equivalente a esas componentes RGB del pixel. Tras esto, situará los rangos para la H y la S de la etiqueta activada alrededor de el valor calculado en el paso anterior, dejando cierta holgura en el rango. Si el resultado no es exactamente el requerido, podemos modificar estos rangos pinchando sobre el histograma como se ha explicado anteriormente, hasta que en la imagen filtrada veamos lo mejor posible aquel objeto cuyo color habíamos seleccionado.

4.3. Segmentación y predicción de trayectorias

La segmentación es una operación de gran importancia en las aplicaciones de procesado de imágenes y de robótica. Para este proyecto nos será útil la segmentación ya que en la RoboCup, los objetos relevantes pueden ser distinguidos por su color. En estas aplicaciones, la ejecución en tiempo real y la robustez de la segmentación son los principales problemas a resolver. La idea básica de la segmentación es agrupar los pixels pertenecientes a un objeto en segmentos. Puede ser utilizada en muchos campos del procesado de imágenes. Por ejemplo, para distinguir en un mapa de rejilla del entorno de un robot móvil las zonas ocupadas por un posible obstáculo o las zonas libres; para la interacción robot-humano, el robot puede utilizar la segmentación para reconocer personas a través del color de la piel y entender órdenes atendiendo al movimiento de la mano humana o incluso analizar la expresión facial de la persona que tenga delante, todo esto a través de la segmentación del color de la piel.

Para realizar la segmentación contamos, en el caso de esta aplicación, con la imagen resultante del filtro de color, que mostrará sobre fondo negro, determinadas áreas de los colores que han pasado el filtro, y que hemos sustituido por un color representante para mayor eficacia. Además existirán sobre la imagen puntos aislados también de estos colores procedentes del ruido ambiental y de pequeñas areas de un color similar al seleccionado y que han superado el filtro, pero que realmente no corresponden a ninguno de los objetos a localizar.

Nuestro objetivo en esta librería es identificar aquellas zonas del color de muestra que cumplan ciertas restricciones sobre la cantidad de pixels que las componen y el grado de agrupación en el que se encuentran. Estas restricciones servirán para descartar los pixels aislados que no pertenecen a ningún objeto relevante. Existen diferentes métodos para implementar la segmentación, pero por razones de coste en tiempo de ejecución usaremos una segmentación basada en histogramas que es muy eficiente.

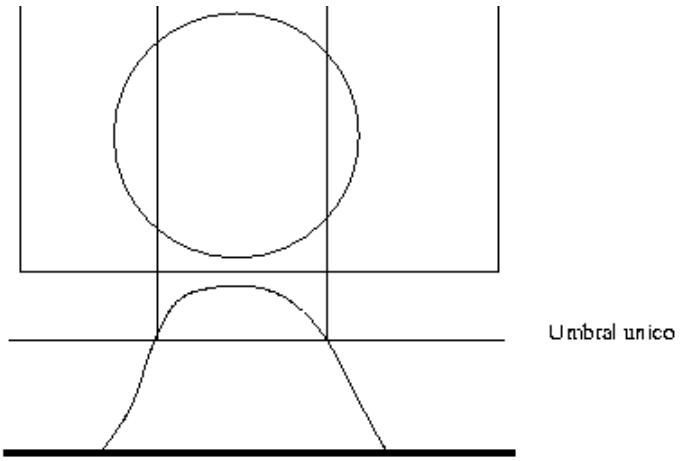

Figura 4.12: Ejemplo de ubral único

Este método para localizar los objetos en la imagen consiste en calcular el histograma del número de pixels del color de muestra existentes tanto por filas (histograma en Y) como por columnas (histograma en X). Combinando ambos histogramas obtendremos una estimación de las zonas de mayor concentración de los puntos buscados. Realizamos un enventanado de la imagen, donde cada ventana, de forma rectangular, encierra uno de los objetos buscados. Para ello, necesitamos obtener los puntos que caracterizan estas ventanas (sus cuatro esquinas), mediante la cual podremos obtener su centro, que coincidirá con el centro del objeto.

Los bordes horizontales y verticales de las ventanas son seleccionados a partir de la información obtenida en los histogramas en X y en Y respectivamente. La elección de estos bordes la realizaremos marcando umbrales en dichos histogramas, de forma que el punto del histograma en el que el número de pixels encontrados supere este umbral, significará la elección de un posible borde de inicio de ventana. A continuación habrá que buscar el borde opuesto de la ventana cuando los valores del histograma dejen de superar el umbral. Esta técnica se puede aplicar de dos formas diferentes:

- Con umbral sencillo. El borde de la ventana se situará justo en la fila o columna de la imagen en la que el número de pixels del color buscado supere este valor umbral. Y el borde opuesto cuando el número de pixels deje de superar el umbral, como se ha explicado en el anterior párrafo (ver figura 4.12).

- Con umbral doble. El método de umbral simple puede provocar que se recorten zonas del objeto como bordes y esquinas que, aunque la suma de sus pixels no supere el umbral, sí deben ser incluídos dentro de la ventana. Para evitar esto, se utilizan dos umbrales por cada borde, siendo el valor de uno ligeramente inferior al del otro. Una vez que se alcanza una fila o columna, según se esté analizando el histograma en X o en Y, en la que su valor en el histograma supera el umbral menor, se sigue recorriendo el histograma. Y si a continuación se supera el umbral mayor, se marca un

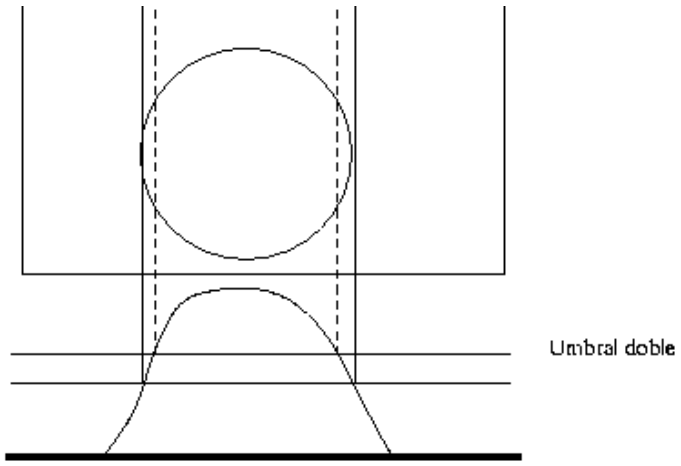

Figura 4.13: Ejemplo de umbral doble

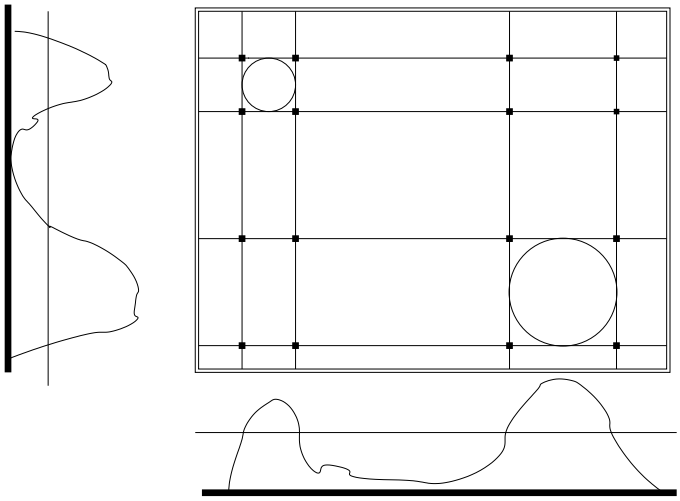

Figura 4.14: Ejemplo de enventanado

borde de la ventana no en el punto donde se supera el umbral mayor, sino donde se supera el umbral menor. De este modo hemos evitado recortar una parte del objeto y nos aseguramos de que no es solamente ruido, pues la densidad es suficientemente alta como para superar el umbral mayor. Para el borde opuesto se realiza la misma operación pero de forma inversa. Se situa el borde de la ventana cuando el valor del histograma es menor que el umbral mínimo (ver figura 4.13).

Con este procedimiento, combinando la información de los histogramas por filas y por columnas, hemos seleccionado las preventanas en las que en unos casos se encontrará el objeto y otros en los que no (figura 4.14. Antes de marcar en la imagen las ventanas obtenidas y calcular su centro, debemos comprobar qué preventanas son válidas y cuáles no. Para ello debemos asegurar que dentro cada preventana hay al menos un número mínimo de píxels del color buscado. En caso contrario, desechamos esa preventana.

Otra funcionalidad implementada en esta librería es la predicción de trayectorias, que consiste en realizar una estimación de la posición de algunos objetos en los instantes siguientes. Algunos de sus posibles usos son:

- Recortar la ventana de búsqueda para tener que filtrar el objeto sólo en una subimagen situada en la posición donde se estima que estará en el instante siguiente.

- Hacer que los robots se puedan anticipar a posiciones futuras de la pelota, mejorando por tanto sus habilidades de juego.

El método elegido es una predicción sencilla basada en la distancia recorrida entre cada imagen. Puesto que la segmentación realiza los cálculos necesarios para obtener el centro de cada objeto localizado, podemos calcular el desplazamiento, en coordenadas X e Y, efectuado por los objetos, entre un instante dado, y el instante siguiente, es decir, entre una imagen y la siguiente. Con esta información podemos estimar que el objeto, en la imagen siguiente va a efectuar un movimiento similar al de la imagen anterior. Por supuesto, en bastantes ocasiones esta predicción será erronea, ya que los robots cambian de dirección y sobretodo la pelota sufre rebotes contra los robots y los bordes del campo, pero puede servir para conseguir que los robots tiendan a seguir a la pelota yendo hacia el punto en el que supuestamente estará en unos instantes después, lo que puede mejorar el rendimiento de los mismos. De lo contrario, los robots se estarían dirigiendo a un punto en el que, cuando alcancen el mismo, la pelota seguramente no estará debido a su rápido movimiento, en cambio, con este método de predicción pueden estarse moviendo hacia el área donde posiblemente se haya desplazado la pelota mientras el robot hacia dicho recorrido.

4.4. Cliente típico

Este cliente utiliza las dos librerías desarrolladas en este proyecto y el interfaz VideoForLinux. Además, el configurador del filtro puede ser utilizado para realizar los ajustes en el fichero de configuración que después utilizará el cliente típico. Consiste en una aplicación que consta de una parte de captura de la imagen de la cámara, el filtrado de la imagen, la segmentación de las ventanas encontradas en ella y la muestra al usuario de ambas imágenes, la real y la procesada con ambas librerías (figura 4.15).

Se trata de un programa de ejemplo de utilización de las librerías, de donde se puede “cortar y pegar” partes del código para reutilizarlas en la aplicación que se desee. Utiliza todos los puntos significativos: captura de imagen, utilización del filtro, utilización de la segmentación y visualización en pantalla.

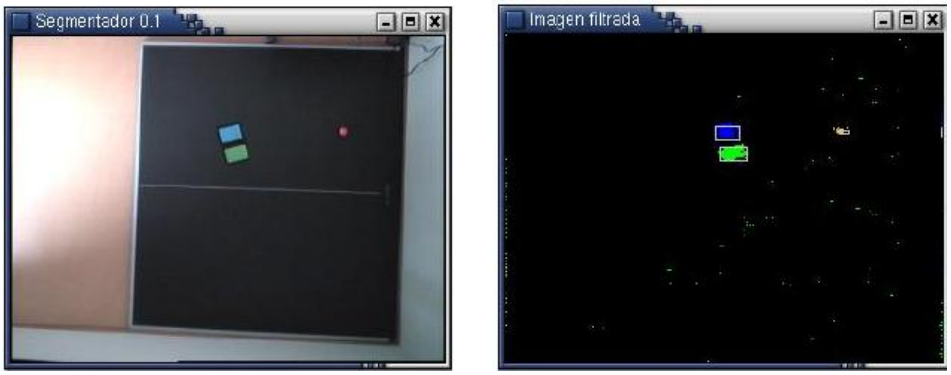

Figura 4.15: Imagen capturada por el cliente (izquierda) e imagen resultante del procesado (izquierda)

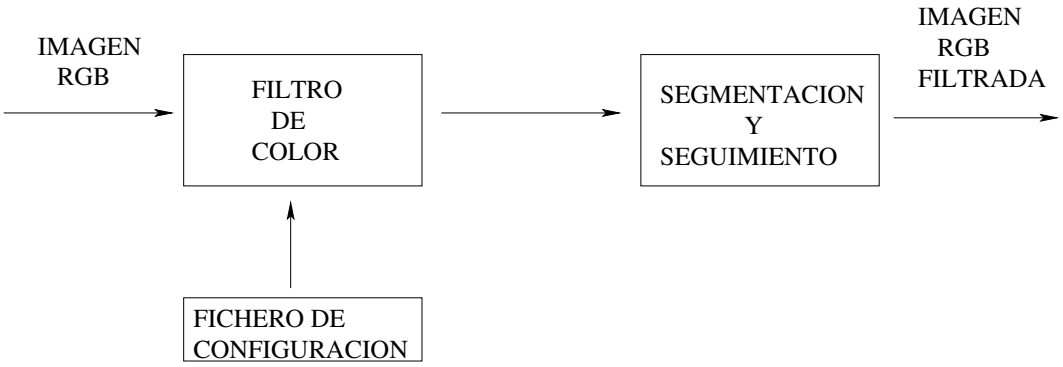

Figura 4.16: Esquema del cliente típico para el filtro y segmentación

Capítulo 5

Conclusiones

El objetivo principal de este proyecto era implementar un filtro de color y su entorno para utilizarlo en la RoboCup. En el presente proyecto se han desarrollado dos librerías y dos aplicaciones orientadas al procesado de imágenes capturadas por una cámara de video conectada al PC. Este procesado, compuesto esencialmente de filtrado de color y segmentación, está relacionado con la visión artificial en el sentido de que sirve a una máquina para interpretar el contenido de las imágenes que captura con la cámara, reconociendo por su color aquellos elementos que el usuario ha configurado previamente.

Biblioteca de filtros de color

Se ha desarrollado un conjunto de filtros que permiten la identificación de colores relevantes en la imagen. Estos filtros se han agrupado en una librería. Los filtros se han implementado en tres espacios de color diferentes (RGB, HSI y LAB) ya que las condiciones de trabajo van a ser variables y cada filtro responde de forma diferente en cada situación. Se pueden utilizar los filtros en los tres espacios de color simultaneamente. En este sentido es una librería muy versátil. Ha sido creada únicamente para participar en la RoboCup, sino como herramienta genérica para trabajar con visión artificial. Este filtro se puede configurar a través de fichero, donde podemos modificar los parámetros deseados escribiéndolos en él.

Recibe como entrada una imagen, que se transforma si es necesario para procesarla. Sólo procesa una imagen a la vez, pero permite trabajar sobre imágenes consecutivas, por ejemplo, provenientes de una cámara. A la salida, el filtro devuelve otra imagen del mismo tamaño y en el mismo formato, pero en la que sólo aparecen los colores representantes de aquellos que hayan superado el filtro.

Respecto a la robustez ante cambios de iluminación, de los tres espacios de color utilizados, el espacio HSI se muestra como el más invariable ante la distinta iluminación de los objetos de la imagen. Aunque esto ya se conocía, ha sido verificado. También se ha implementado el filtro RGB debido a la coincidencia con el formato de captura y visualización, ya que otro factor importante en el procesado de imágenes es el tiempo de proceso. Por tanto, en determinadas condiciones, el uso del filtro RGB dará mayor

vivacidad al sistema.

El modelo Lab es un modelo de color tan complejo como el HSI y difícil de representar gráficamente. Ha sido elegido, al igual que el HSI, por su invarianza ante cambios de iluminación.

Herramienta de configuración

Se ha implemetando una herramienta que permite configurar y realizar cambios en el filtro de forma más cómoda. Recordemos que la otra forma de configurar el filtro consiste en modificar con un editor de texto el fichero de configuración, pero su contenido numérico puede ser muy engorroso y poco orientativo para el usuario y requeriría muchas pruebas hasta alcanzar una configuración óptima. Por eso se ha creado este configurador, en el cual, el usuario trabaja directamente con los colores que ve en la pantalla. Su principal ventaja consiste en que para el usuario es más intuitivo seleccionar directamente aquellos colores que quiere filtrar, sin tener que conocer los valores numéricos de su representación en los espacios de color RGB y HSI. En el caso del espacio LAB, dado que es difícil representarlo, se ha mantenido la configuración numérica, pero sus valores se modifican de modo sencillo con un dial. Esto no es un gran inconveniente porque la aplicación tiene otras ventajas, que veremos a continuación, que agilizan la configuración.

Mediante la interfaz gráfica de esta herramienta se presenta una abstracción que ofrece, del lado del usuario, la representación de los espacios de color con los que puede trabajar. Sobre esta representación, se inscriben los colores que van a pasar el filtro en forma de sectores: "quesitos" para el HSI y rectángulos para el RGB. Esta es una forma muy sencilla de permitir que el usuario sepa con sólo mirar el interfaz qué colores está filtrando exactamente. Y si este observa que estos sectores no engloban correctamente el color elegido, podrá redimensionarlos y colocarlos exactamente donde lo crea necesario arrastrándolos con el ratón.

Otra ventaja de la implementación de esta herramienta es que muestra tanto la imagen real como la imagen ya filtrada. Esto es muy útil para comprobar al momento si la configuración elegida es correcta. Puesto que puede filtrar varios colores a la vez, en diferentes espacios, para cada color se elige un color representante que será el que se visualice en la imagen de salida, en sustitución del color original que ha sido filtrado.

Permite además la selección del color de muestra. El usuario sólo tiene que pinchar con el ratón sobre la imagen real del objeto cuyo color quiere filtrar y el filtro emplazará la etiqueta que esté configurando englobando el color que haya pinchado. De este modo, en pocos instantes se puede tener el filtro trabajando sobre cualquier color que aparezca en la imagen.

Biblioteca de segmentación

Uno de los objetivos de este proyecto era desarrollar un sistema que permitiese interpretar, en la imagen resultante del filtro, las zonas de determinado color, que se corresponderían con objetos presentados ante la cámara. Así, podría ser utilizado en cualquier otro programa como base para el seguimiento de objetos. En nuestro caso, también es necesario para el equipo de fútbol robótico que debe conocer en todo momento la posición de los robots a través de una cámara cenital. Esto se consigue mediante un sistema de segmentación como el que se ha implementado en la librería correspondiente. Se ha programado una serie de funciones que permiten, dada una imagen, buscar y enventanar aquellas zonas de la imagen formadas por pixels consecutivos del mismo color. Este tipo de segmentación ha sido implementada usando histogramas. Una vez hecho esto, la librería permite calcular las coordenadas del centro de dichas áreas. Además se ha programado la predicción del movimiento de los objetos, mediante una estimación sencilla, que puede resultar ventajosa a la hora de organizar el juego de los robots.

Cliente típico

Además de los anteriores elementos, se ha desarrollado un programa de ejemplo que utiliza todos ellos, sirviendo de explicación del código para quienes realicen futuros desarrollos basados en el filtro. Realiza el procesado completo de la imagen, incluyendo: lectura de la imagen proveniente de la captura, filtrado de la imagen según los parámetros que contiene el fichero de configuración y creación de una nueva imagen que sólo muestre la información que no haya sido descartada por el filtro, segmentación de cada uno de los colores que aparecen en la imagen filtrada y uso de las técnicas de seguimiento de objetos implementadas en la librería correspondiente.

5.1. Líneas futuras

La evolución natural de este proyecto dentro del Grupo, consiste en incluírlo como parte del sistema para participar en la RoboCup. De esta tarea se está ocupando actualmente Marta Martín en su PFC [Martín03], que trata de resolver el seguimiento de la pelota desde la cámara cenital, relizando un filtrado de color como el que trata este proyecto y enviando las órdenes pertinentes desde el PC al robot vía radio, para hacer que éste persiga la pelota.

Otra posible línea futura es implementar el filtro en hardware, diseñando y fabricando una tarjeta PCI para el PC. Esto aceleraría en gran medida su velocidad de funcionamiento al no depender de la velocidad del procesador principal del equipo en el que funcione.

También se pueden incorporar nuevas funciones a la librería de manipulación de imágenes: filtro de bordes, filtro de esquinas, etc.

Bibliografía

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

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

[Gómez02] GOMEZ, V.: *"Comportamiento sigue pared en un robot con visión local."*, Proyecto Fin de Carrera, Universidad Rey Juan Carlos, 2002.

[Agüero02] Agüero, C.: *"Protocolo de Encaminamiento para Redes Adhoc."*, Proyecto Fin de Carrera, Universidad Rey Juan Carlos, 2002.

[Martín03] Martín, M.: *"Comportamiento Sigue Pelota cenital"*, Proyecto Fin de Carrera, Universidad Rey Juan Carlos, 2003.

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

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

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