Proyecto Final de Carrera

Realidad Aumentada: ARToolKit para animación de personajes

Manuel Ibáñez Herrero

Dirigido por:

Manuel Agustí i Melchor
Gabriela Andreu García

Departamento de Informática de Sistemas y Compudores
Escuela Técnica Superior de Informatica Aplicada
Universidad Politécnica de Valencia

Índice

ANEXOS

Primera parte

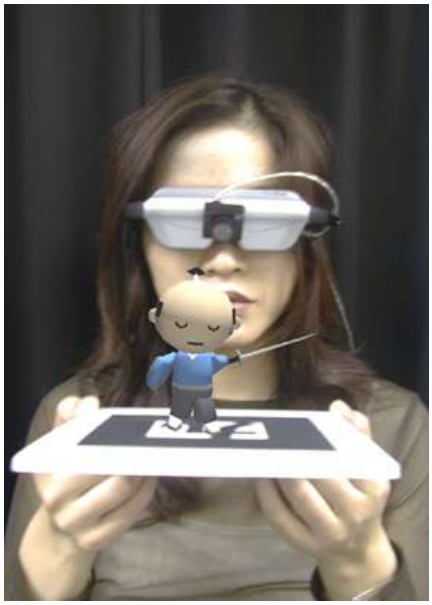
ARToolkit y Realidad Aumentada

1.Realidad Aumentada

1.1¿Qué es?

La realidad aumentada es un sistema de interacción que toma como entrada la información que proviene del mundo real y genera información de salida (tal como objetos, imágenes, texto, etc.) que se superpone en tiempo real sobre la percepción que el usuario tiene del mundo real, consiguiendo así un aumento en el conocimiento que el usuario tiene sobre los objetos de su entorno.

Supone una inmersión, por parte del usuario, en un mundo que resulta de la "unión" entre el mundo real y el mundo virtual, ya que el usuario de este tipo de aplicaciónes, podrá ver (a través de una cámara, o de dispositivos especiales de visión) objetos generados por ordenador que se integran en el mundo real.



1.2 Realidad Aumentada VS Realidad Virtual

La realidad virtual pretende la inmersión del usuario en un mundo totalmente virtual, donde todo aquello que percibe ha sido generado por ordenador. El usuario se encuentra en un mundo distinto, aislado del mundo real, rodeado de objetos virtuales que no existen en la realidad, pero puede interactuar con ellos como si de verdad existieran.

En cambio, la realidad aumentada no pretende aislar al usuario del mundo real, sino complementar éste mediante objetos virtuales e imágenes generadas por ordenador. El usuario se encuentra inmerso en un mundo que tiene a la vez elementos virtuales y elementos reales con los que puede interactuar.

2. Introducción a ARToolKit

2.1 ¿Qué es?

ArToolKit es un conjunto de librerías para C/C++ que sirven para la creación de aplicaciones de realidad aumentada. Para ello proporciona una serie de funciones para la captura de vídeo y para la búsqueda de ciertos patrones, en las imágenes capturadas, mediante técnicas de visión por computador. También proporciona una serie de ejemplos y utilidades de gran ayuda al programador que quiera realizar este tipo de aplicaciones.

2.2 ¿Cómo funciona?

Algo muy importante en las aplicaciones de realidad aumentada, es la necesidad de calcular el punto de vista de la cámara, para así poder realizar las operaciones necesarias sobre los objetos virtuales, para que estos se integren correctamente en el mundo real. Es decir, si queremos mostrar objetos virtuales, de modo que el usuario realmente se crea que existen en el mundo real, tendremos que realizar transformaciones sobre esos objetos de modo que el usuario los vea (a través de la cámara o dispositivo de captura utilizado) en la posición, tamaño, orientación e iluminación, en que esos objetos serían percibidos por el usuario en el mundo real en caso de que realmente estuvieran allí.

Para ello se utilizan unas plantillas de forma cuadrada, que se componen de un cuadrado negro con un cuadrado blanco cuatro veces más pequeño en su centro, y un dibujo sencillo en el interior del cuadrado blanco. La aplicación, utilizando las funciones y utilidades proporcionadas por ARToolKit, será capaz de detectar una de estas plantillas en las imágenes de vídeo capturadas.

Aquí se muestra un ejemplo de plantilla:

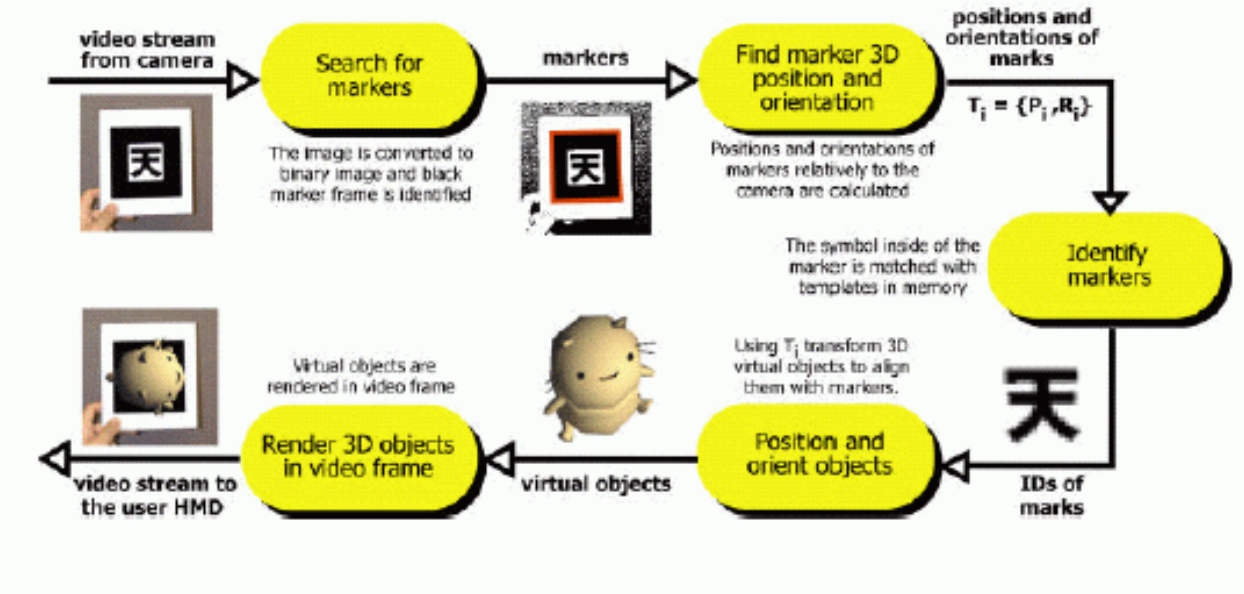
Una vez detectada una plantilla en una imagen, estudiando la orientación, posición y tamaño de la plantilla, la aplicación es capaz de calcular la posición y orientación relativa de la cámara respecto a la plantilla, y usando esta información podrá pasar a dibujar el objeto correspondiente sobre la imagen capturada mediante librerias externas a ARToolKit (por ejemplo GLUT y OpenGL), de modo que el objeto aparezca sobre la plantilla en la posición, orientación y tamaño correspondiente al punto de vista de la cámara, siempre que el programador de la aplicación así lo haya decidido, pues las posibilidades son muchas y pudiera ser que una vez obtenida esta información el programador decidiese utilizarla de otra forma, hacer otras operaciones distintas, etc.

El funcionamiento básico de una aplicación de ARToolkit es el siguiente:

- Primero se captura un fotograma del mundo real mediante la cámara.
- A continuación la imagen se umbraliza con cierto valor del umbral (threshold), de forma que los pixels cuya intensidad supere el valor del umbral son transformados en pixels de color negro. El resto se transforman en píxels blancos.
- Se buscan y encuentran todos los marcos negros como los de la plantilla existentes en la imagen (en realidad al umbralizar la imagen el marco aparece blanco y el cuadrado blanco aparece negro).

- Se compara el interior del marco con las plantillas de las que se tiene información almacenada.
- Si la forma de la plantilla analizada y la plantilla almacenada coincide, se utiliza la información de tamaño y orientación de la plantilla almacenada para compararla con la plantilla que se ha detectado y así poder calcular la posición y orientación relativas de la cámara a la plantilla, y se guarda en una matriz.
- Se utiliza esta matriz para establecer la posición y orientación de la cámara virtual (transformación de la vista), lo que equivale a una transformación de las coordenadas del objeto a dibujar.
- Al haber puesto la cámara virtual en la misma posición y orientación que la camara real, el objeto virtual se dibuja sobre la plantilla, se renderiza y se muestra la imagen resultante, que contiene la imagen del mundo real y el objeto virtual superpuesto, alineado sobre la plantilla.
- Se realiza el mismo proceso con los siguientes fotogramas.

El siguiente diagrama muestra el funcionamiento que se acaba de describir:



2.3 Instalación

2.3.1 Instalación en Windows

Software necesario:

- ARToolkit 2.70.1 y DSVideoLib-0.0.4-win32: http://sf.net/projects/artoolkit
- GLUT: http://www.opengl.org/resources/libraries/glut.html
- DirectX 9.0b SDK o superior (si se va a usar VC6 la version del directX SDK no puede ser otra que la 9.0b): http://msdn.microsoft.com/library/default.asp?url=/downloads/list/directx.asp
- Extras del directX 9.0 SDK para C++: http://www.microsoft.com/downloads/details.aspx?FamilyID=bc7ddedd-af62-493d-8055-5e57bab71e1a

Pasos de instalación usando Microsoft Visual Studio 6:

1. Instalar el DirectX 9.0b SDK

2. Descomprimir ARToolKit (version 2.70.1) en la carpeta ARToolkit
3. Descomprimir DSVideoLib en ARToolKit\DSVideoLib
4. Copiar los ficheros DSVideoLib.dll y DSVideoLibd.dll desde ARToolKit\DSVideoLib\bin.vc60 al directorio bin de ARToolkit
5. Ejecutar el script register_filter.bat del directorio ARToolKit\DSVideoLib\bin.vc60
6. Descomprimir e instalar las librerías GLUT, para ello, se situarán (manualmente) los ficheros *.h y *.lib en los directorios ARToolKit\include\GL y ARToolKit\lib, respectivamente. Los ficheros *.dll se colocan o bien en el directorio system32 de windows o en el directorio en que van a estar los ejecutables (directorio ARToolKit\bin).
7. Ejecutar ARToolKit\Configure.win32.bat con lo que se creará el fichero ARToolKit\include\AR\config.h
8. Descomprimir en el directorio dx9sdkcp el fichero con los extras del directX 9 SDK para C++ (dx9sdkcp.exe) y copiar el directorio dx9sdkcp\SDK (c++)\include a ARToolKit\directX\
9. Copiar el directorio dx9sdkcp\SDK (C++)\Samples\C++\DirectShow\BaseClasses a ARToolKit\directX\ (una vez se haya copiado este directorio y el directorio dx9sdkcp\SDK (c++)\include del punto anterior, si se desea puede borrarse el directorio dx9sdkcp ya que no lo necesitaremos más).
10. Abrir el proyecto de VS6 (fichero .dsw) con microsoft visual studio
11. Configurar los directorios include y lib en el microsoft visual studio (Opción “Options” del menú “Tools”):

    Include:

    - Añadir el directorio ARToolKit\include
    - Añadir el directorio ARToolKit\DsVideoLib\SRC
    - Añadir el directorio ARToolkit\directX\BaseClasses
    - Añadir el direcorio ARToolKit\directX y situarlo el primero de la lista, es muy importante que esté el primero

    Lib:

    - Añadir a los directorios lib el directorio ARToolKit\lib
12. Compilar y construir las clases del proyecto: Si todo ha ido bien, se habrán generado los ficheros de ejemplo en el directorio ARToolKit\bin

Nota: En caso de tener tarjeta de TV, puede ser necesario desinstalarla o deshabilitarla, ya que sino al ejecutar los ejecutables podría aparecer la señal de TV en lugar de la de la camara.

2.3.2 Instalación en Linux

2.4 Ejemplos

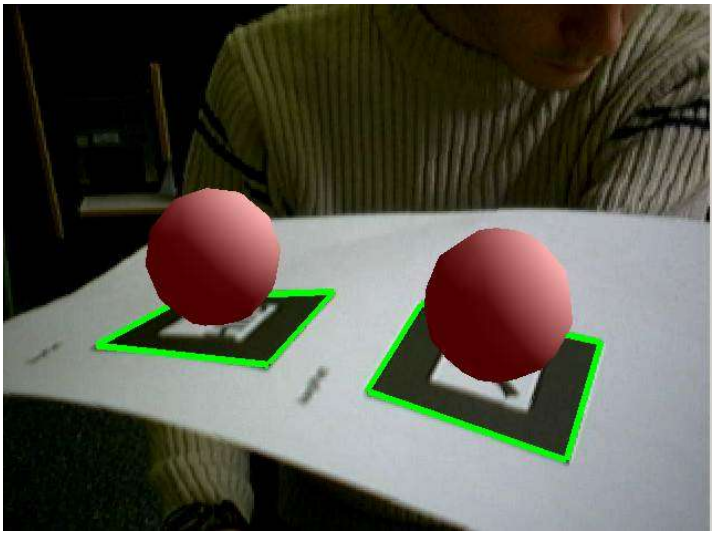
ARToolKit 2.70.1 viene con los siguientes ejemplos:

calib_camera2d: Sirve para medir la distancia focal de la cámara y otros parámetros.

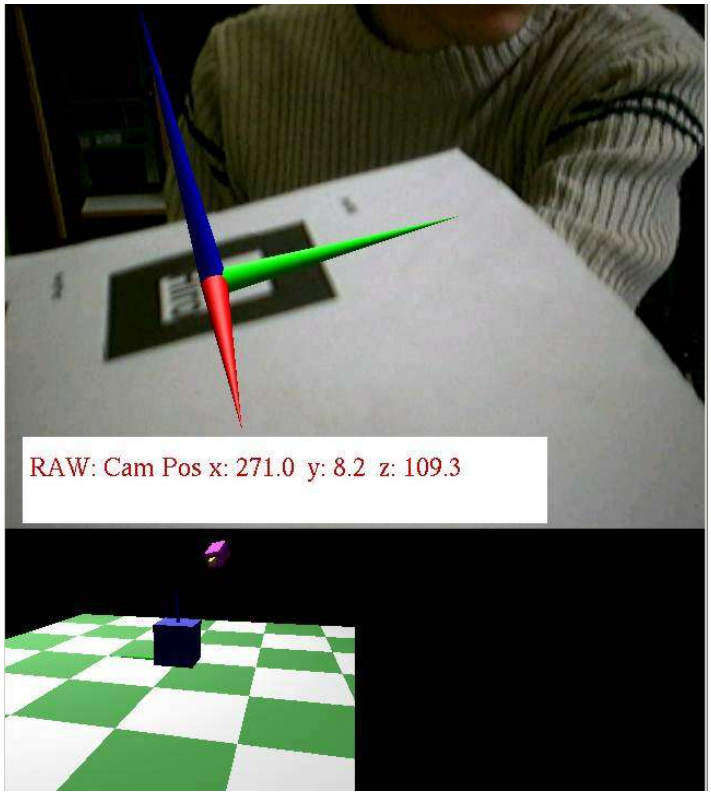
calib_cparamd: Sirve para medir las propiedades de la cámara.

calib_Distorsiond: Para medir la distorsion de la cámara.

collideTestd: Detecta colisiones entre plantillas. Marca las plantillas detectadas con líneas rojas y cuando están a cierta distancia una de otra, las marca en verde.

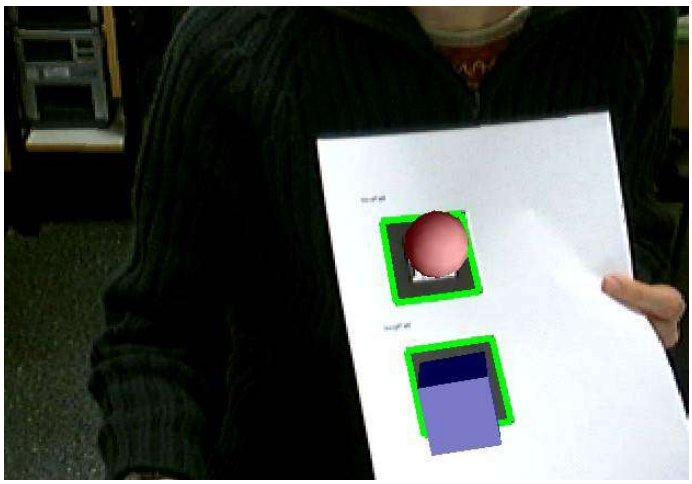


exviewd: Calcula y muestra la posición de la cámara a partir de la plantilla “Hiro”. Presionando 'd' muestra en una esquina al imagen umbralizada. Presionando 't' deja ajustar el valor del umbral.



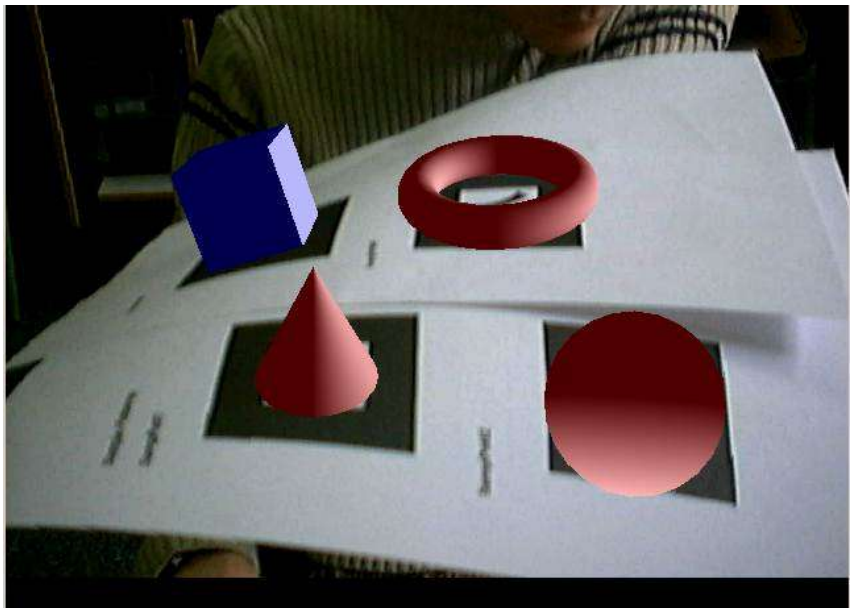
graphicsTestd: Simplemente muestra una tetera girando para hacer una prueba de los gráficos.

loadMultipled: Reconoce las plantillas “Hiro” y el kanji “Hito”, y dibuja una esfera sobre la primera y un cubo sobre la segunda. Presionando 'd' muestra en una esquina la imagen umbralizada y con 't' permite ajustar el valor del umbral.

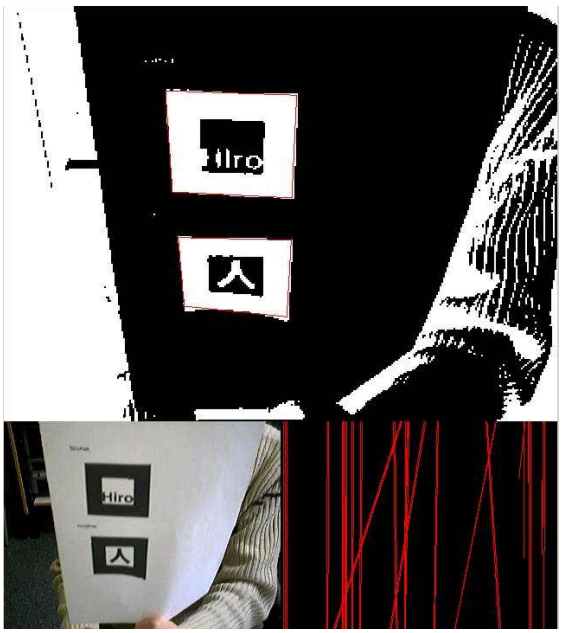


mkpattd: Sirve para crear nuevas plantillas a partir de una imagen de la plantilla capturada por la cámara. Presionando 't' se puede ajustar umbral.

modeTestd: Dibuja un cubo sobre la plantilla “Hiro”, un toroide sobre el kanji “Hito”, un cono sobre “samppatt1”, y una esfera sobre “samppatt2”. Presionando 'd' muestra en una esquina la imagen umbralizada. Y con 't' permite ajustar el valor del umbral.



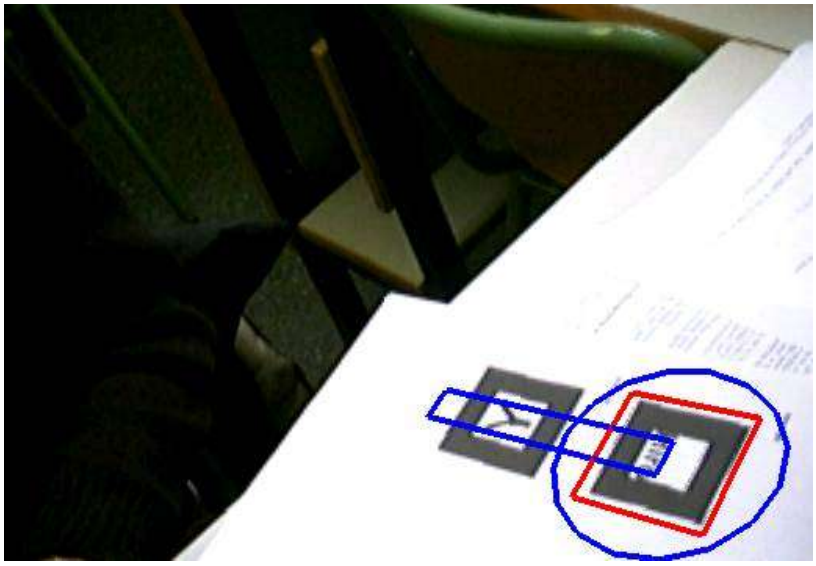
multid: Muestra la señal de vídeo de la cámara. Presionando 'd' muestra en una ventana grande la imagen umbralizada y con 't' deja ajustar el valor del umbral. Si encuentra alguna de las plantillas, las marca en rojo sobre la imagen umbralizada.



opticald: Sirve para calibrar dispositivos de visión. Dibuja un cubo sobre “hiro”, un toroide sobre el kanji "Hito", un cono sobre “samppatt1”, y una esfera sobre “samppatt2”. Presionando 'd' muestra en una esquina la imagen umbralizada. Con 't' deja ajustar el valor del umbral

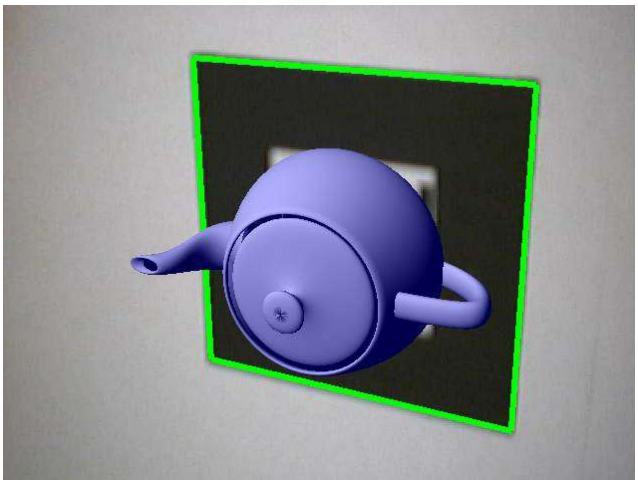
paddleInteraction: Dibuja una raqueta de paddle sobre “hiro”. Presionando 'd' muestra en una esquina la imagen umbralizada. Con 't' permite ajustar el valor del umbral utilizado.

paddleTestd: Dibuja una raqueta de paddle sobre “hiro”. Presionando 'd' muestra en una esquina la imagen umbralizada y pulsando la tecla 't' se puede ajustar el valor del umbral.

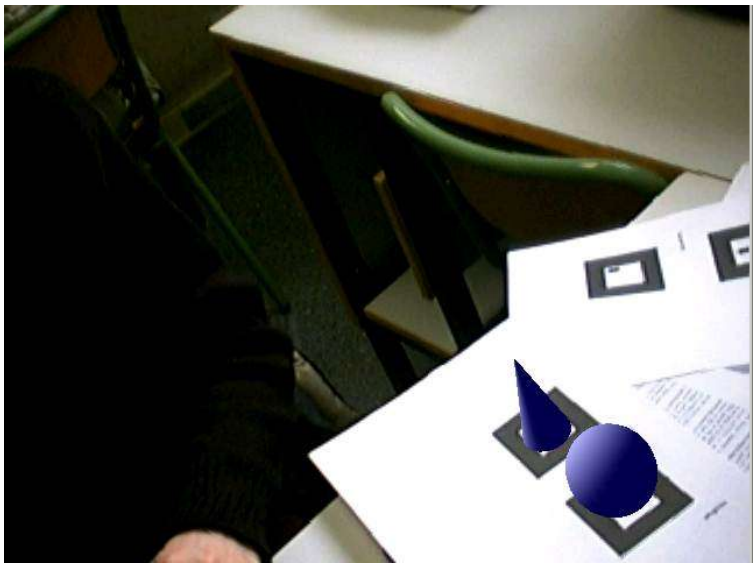


rangeTestd: Interacción entre la cámara y las plantillas. Dibuja una tetera sobre la plantilla “Hiro”, conforme la marca se acerca a la cámara el objeto aparece más pequeño.

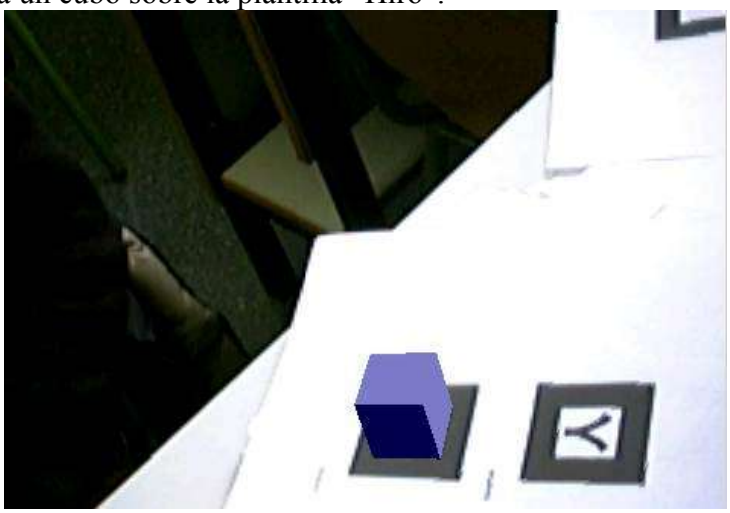




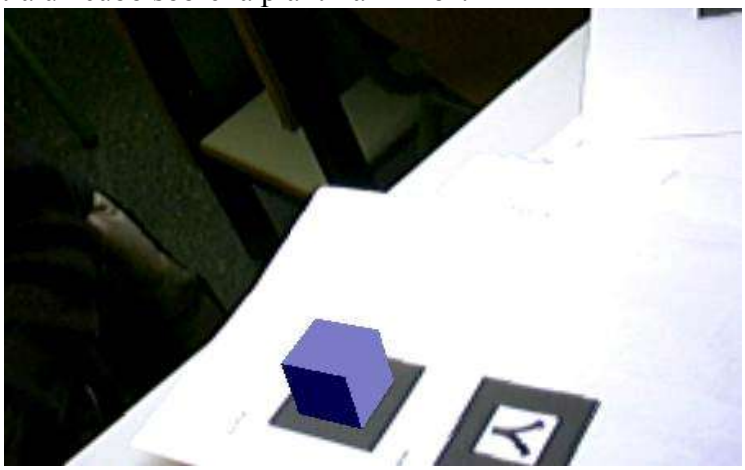
relationTestd: Muestra un cono sobre el kanji “hito” y una esfera sobre “Hiro”, y además cuando encuentra las dos plantillas, muestra por consola una matriz 3x4 que refleja la diferencia de posición y rotación entre una plantilla y la otra.



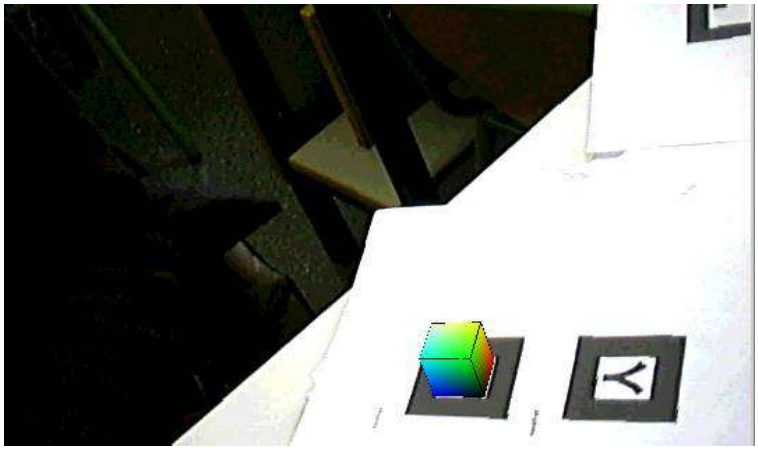
simpleTestd: Muestra un cubo sobre la plantilla “Hiro”.



simpleTest2d: Muestra un cubo sobre la plantilla “Hiro”.



simpleLited: Muestra un cubo con una textura formada por degradados de color sobre la plantilla “Hiro”.



twoViewd: Para dispositivos con vision estereoscópica.

videoTestd: Muestra la señal de vídeo de la cámara.

Segunda parte

El proyecto

1. Introducción al proyecto

1.1¿Qué queríamos conseguir?

Como se vio en la primera parte, las librerias ARToolKit permiten realizar aplicaciones de Realidad Aumentada basadas en el seguimiento de unas plantillas con unas características muy concretas. Las plantillas tienen que estar formadas por un cuadrado negro dentro del cual existe otro cuadrado blanco, y dentro de este último cuadrado se encuentra la forma que el programa debe reconocer. A parte de esto, los programas realizados de esta forma no reconocen plantillas de color, ya que la información extraida de la imagen capturada es tratada como si fuera una imagen en escala de grises.

Nuestra idea era dejar de lado estas plantillas tan restrictivas y con características tan específicas y sustituirlas por otros patrones más naturales como pudieran ser un bolígrafo, una mano, un dedo o incluso una cara. Utilizándolos ya no solo para dibujar un objeto virtual sobre ellos, sino como dispositivos de interacción, dispositivos apuntadores que permitieran la interacción con un programa, de forma parecida a como lo hacen el ratón o el joystick en las aplicaciones convencionales.

1.2¿Qué hemos conseguido?

Se han añadido algunas funciones a las librerias de ARToolKit que amplían la funcionalidad de éstas. A través de estas funciones, es posible trabajar con unos patrones muy distintos a aquellas plantillas cuadradas y negras.

Se ha eliminado la restriccion de forma, los patrones utilizados ya no tienen porque ser cuadrados, pueden tener cualquier forma. El usuario es libre de elegir la forma que prefiera ya que estas funciones nunca comprueban la forma del objeto. Esto puede ser un problema si en las aplicaciones se desea obtener información sobre la orientación y posición de los patrones en el espacio 3D, ya que al no controlarse la forma no se ofrece ninguna posibilidad de obtener dicha información. No obstante, no es nuestro objetivo utilizar información sobre la orientación y posición en un espacio 3D, nos es suficiente con la información de posición en un espacio 2D y para ello no es necesario restringir la forma de los patrones.

Los patrones utilizados ya no tienen porque ser negros o grises, de hecho es recomendable que no lo sean. Esto es debido al hecho de que estas funciones basan su funcionamiento en la información de color de los patrones. Es el usuario el que elige un patrón del color que desee, luego utilizando una utilidad de configuración, puede crear un fichero de configuración que ayudará a los programas a reconocer el patrón elegido según sea su color. A partir de entonces, puesto que no hay restricción de forma, solo de color, cualquier objeto que se encuentre dentro de los rangos de color seleccionados, será identificado por el programa como el patrón a seguir.

Con el fin de mostrar como los patrones de este tipo pueden ser utilizados por las aplicaciones como dispositivos de interacción, se han creado algunos programas de ejemplo en los que el usuario puede escoger su propio patrón para utilizarlo como dispositivo de interacción y realizar con él algunas de las funciones básicas que podría realizar con un ratón o joystick. Estos programas de ejemplo se verán más adelante.

1.3 Limitaciones

Aunque en un principio el objetivo fue poder reconocer objetos cotidianos que hicieran de dispositivo apuntador, lo cual requeriría no solo del reconocimiento de información de color de los objetos, sino también de informacion sobre los objetos del mundo real que deben buscarse. Debido

a la complejidad de implementar las operaciones necesarias para realizar un reconocimiento de forma eficaz, se ha utilizado tan solo la información de color, mucho más fácil y eficiente de tratar, para identificar a los objetos.

Como se dijo anteriormente, el hecho de no incluir un análisis de la forma de los objetos, hace imposible obtener de ellos información sobre la posición y orientación del objeto en el espacio 3D. En nuestro caso, puesto que queremos usar los patrones como un dispositivo apuntador, solo nos interesa saber en que coordenadas X e Y de la pantalla se proyecta su imagen.

Puesto que los objetos solo se distinguen por su color, es posible que las aplicaciones se encuentren con que ese color se repite no solo en el patrón, sino en otros objetos del entorno, en esta situación el programa no funcionaría correctamente, ya que no sabría distinguir cual es el patrón. El usuario es responsable de elegir los patrones y las condiciones de trabajo tales que garanticen que los colores del patrón no se repiten en el entorno de trabajo, para evitar que sean erróneamente identificados como patrones por las aplicaciones.

Además, el hecho de tratar con información de color, aporta ciertas dificultades. Por ejemplo, las condiciones de iluminación pueden afectar al funcionamiento de los programas, por el hecho de cambiar el color que se percibe de los objetos. Así mismo, es posible que un patrón que el programa detecta perfectamente en una posición, deje de detectarse al moverse, debido a los cambios de iluminación que se producen sobre él por el mero hecho de cambiar de posición o de orientación, o que unas partes del objeto sean detectadas mientras que otras partes más sombreadas, o mas iluminadas no. Con el fin de reducir las consecuencias de este problema, la información de color es convertida al espacio de color HSV en lugar de RGB, debido a que HSV es un espacio mucho más fácil de controlar. Sabemos que en HSV un cambio en la iluminación afectará sobretodo al valor de V, es decir a la componente de brillo, mientras que afectará menos a la saturación y al tono, en cambio en RGB, un cambio la iluminación puede afectar a las tres componentes R, G y B de una forma mucho más dificil de calcular. Además se ha creado una utilidad de configuración que permite ajustar los rangos de valores de color HSV que se buscarán en la imagen, de forma que estos rangos incluyan las tonalidades que el objeto presenta al sombrearse o iluminarse.

2. Instalación y ejecución

2.1 Instalación en windows

Se detallarán los pasos para dos tipos de instalación para Windows:

1. Una instalación en modo ejecución para un usuario común, que tan solo permite acceder a los ejecutables, pudiendo así utilizar los programas pero no modificarlos. Esta instalación está aconsejada para usuarios sin conocimientos de programación, o que no deseen realizar modificaciones ni analizar el código.
2. Una instalación en modo desarrollo para usuarios avanzados, que permite acceder al código de las librerías y aplicaciones desarrolladas, pudiendo realizarse modificaciones sobre cualquiera de ellos. Se requieren conocimientos de programación en C++.

En ambos casos, para facilitar la instalación, se han incluido en un fichero comprimido todos los componentes necesarios, evitando así que el usuario tenga que adquirirlos e instalarlos por su cuenta. No obstante para el primer tipo de instalación, se asume que el usuario tiene instalado el Microsoft Visual Studio C++.

2.1.1 Instalación en modo ejecución:

La instalación en modo ejecución es muy fácil, simplemente siga estos dos pasos:

- Descomprimir el fichero ????.zip en un directorio. En adelante cuando se escriba una ruta se hará considerando tal directorio como directorio raiz, abreviándose mediante el símbolo "\", de forma que al escribir "\miDirectorio\miFichero" nos referimos al fichero, de nombre "miFichero", que se encuentra dentro de un directorio, llamado "miDirectorio", que a su vez está situado dentro del directorio en el que se descomprimió el fichero comprimido.
- En el directorio \bin encontrará los ejemplos de ARToolKit, junto con nuestros propios ejemplos listos para ejecutar.

2.1.2 Instalación en modo desarrollo

Los pasos a realizar para realizar la instalacion en modo desarrollo para Windows con MSVSC++ son:

- Descomprimir el fichero ????.zip en un directorio. En adelante cuando se escriba una ruta se hará considerando tal directorio como directorio raiz, abreviándose mediante el símbolo "\", de forma que al escribir "\miDirectorio\miFichero" nos referimos al fichero, de nombre "miFichero", que se encuentra dentro de un directorio, llamado "miDirectorio", que a su vez está situado dentro del directorio en el que se descomprimió el fichero comprimido.
- Entrar al directorio \DsVideoLib\bin.vc60 si la versión de MSVSC++ utilizada es la 6.0, o al directorio \DsVideoLib\bin.vc70 si se utiliza la versión 7.0. Una vez dentro se deben copiar los dos ficheros con extensión dll al directorio \bin, o bien al directorio system de windows. Hecho esto, hay que ejecutar el fichero register_filter.bat.
- Abra ahora el espacio de trabajo \ARToolKit.dsw con MSVSC++. Vaya a herramientas/opciones (tools/options en la versión en inglés) y seleccione la pestaña directorios (directories). Ahora añadiremos los directorios de "includes" necesarios y posteriormente los de librerías o "lib".
- Seleccione "Include files" y añada los siguientes directorios[1]:

  \include
  \dsvideolib\src
  \directx\baseclasses
  \directc\include
  \allegro\include (solo es necesario si desea compilar el ejemplo sample2)

- Seleccione ahora "lib files" y añada los siguientes directorios:

  \lib
  \allegro\lib (necesario solo para compilar el ejemplo sample2)

- Ahora está todo listo para la construcción de las librerías y aplicaciones. Elija la opción de reconstruir todo (rebuild all). Si esto da problemas, puede probar a construir cada proyecto por separado, empezando por los que empiezan por libAR, ya que estos son los que implementan

1 Recuerde que el simbolo "\" que antecede a cada ruta es una abreviación, y que por tanto debe sustituirse por la ruta completa del directorio en el que se descomprimió el fichero comprimido. Además es importante que el directorio "\directc\include" sea el primero de la lista, para ello súbalo con el botón de la flecha que apunta hacia arriba hasta que se encuentre en la primera posición).

las librerías necesarias para todos los demás.

- Si todo ha ido bien, en el directorio \bin encontrará los ejemplos de ARToolKit, junto con nuestros propios ejemplos listos para ejecutar.

2.2 Instalación en Linux

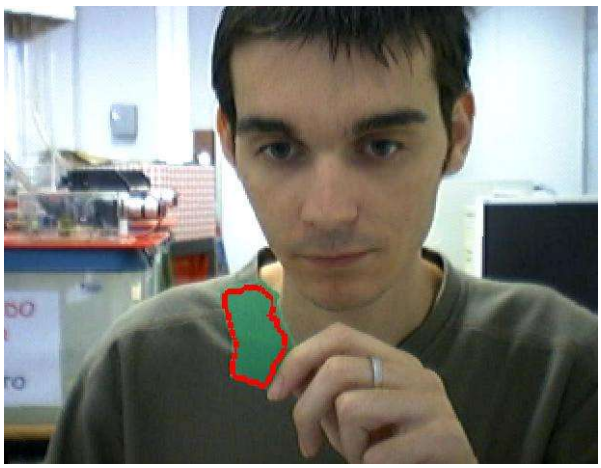
Para la instalación en Linux, solo se explica una isntalación en modo desarrollo.

3. Aplicaciones de ejemplo y utilidades

En este punto, se describe el funcionamiento de los ejemplos y utilidades desarrollados:

Configuration: Debe utilizar esta aplicación antes que las demás, ya que le ayudará a elegir un patrón que hará de puntero en los otros programas. Debe seguir los siguientes pasos para entrenar al sistema a reconocer el objetivo a seguir:

1. Sitúe el patrón elegido delante de la cámara, intente que sea de un color que no se repita en el entorno de trabajo.
2. Cuando vea su objetivo en la pantalla, haga *click* sobre él con el botón derecho del ratón.
3. Realice varios clicks sobre la superficie del objetivo a seguir con el boton izquierdo del ratón, hasta que la superficie de este aparezca totalmente bordeada. Debe hacer los clicks en aquellas zonas que aun no aparecen bordeadas, de esta forma se añadirán a la selección de color.
4. Para evitar problemas con la iluminación, mueva o gire el patrón delante de la cámara a distintas posiciones del entorno de trabajo, verá como en ocasiones la totalidad o parte del patrón deja de estar bordeado, simplemente haga *click* sobre las zonas no bordeadas para que se añadan. Continúe de esta forma hasta que este satisfecho con el seguimiento realizado. Si esto no funciona o comienzan a seleccionarse elementos que no forman parte del patrón, repita desde el paso 2.
5. También puede ajustar manualmente los rangos de H (hue o tono), S (saturation o saturación) y V (value o valor de brillo) que se detectarán, pulsando respectivamente las teclas 'h', 's' y 'b'. Debe escoger valores entre 0 y 360 para H, y entre 0 y 100 para S y V.
6. Puede ajustar un umbral pulsando 'u'. Cuanto mayor sea el umbral, mayor será la desviación permitida respecto de los valores de HSV elegidos.
7. Cuando termine pulse 'g' y confirme escribiendo “s” en la consola de texto para guardar la configuracion en el fichero conf.con.



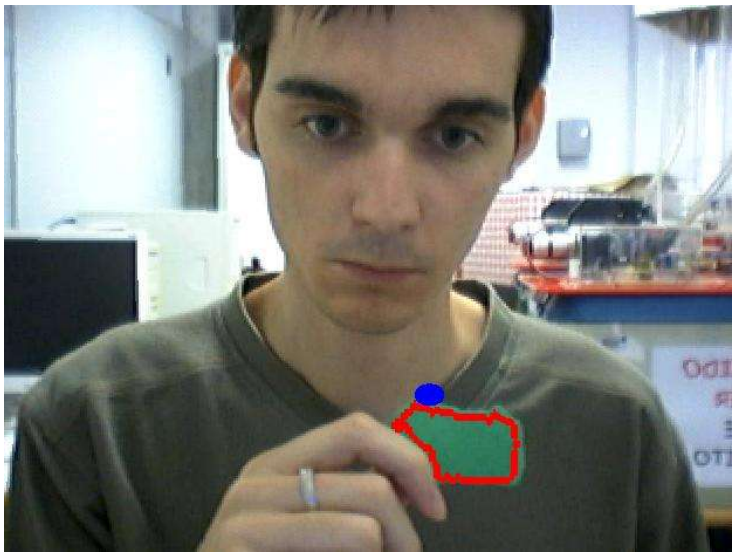
En estas dos imágenes puede verse como se han detectado, por separado, dos patrones diferentes utilizando este

programa.

El fichero de configuración también puede editarse manualmente utilizando un editor de texto. El formato del fichero conf.con es el siguiente:

Umbral <-- La tolerancia permitida, solo aparece una vez en la primera línea del fichero
# <-- carácter separador '#'
minH maxH <-- Los valores mínimo y máximo de H, separados por un espacio
minS maxS <-- Los valores mínimo y máximo de S, separados por un espacio
minV maxV <-- Los valores mínimo y máximo de V, separados por un espacio
- <-- carácter que señala el final del fichero

Sample: Este ejemplo es una especie de juego en el que una pelotita se mueve por la pantalla y rebota cuando colisiona con el patrón seleccionado con la utilidad de configuración. Sitúe su patrón delante de la cámara y compruebe que aparece bordeado, si no es así, o si al moverlo deja de estar bordeado total o parcialmente, pruebe a ejecutar de nuevo el programa **Configuration**. Mueva el patrón y observe como la pelotita rebota cuando se encuentra con él.

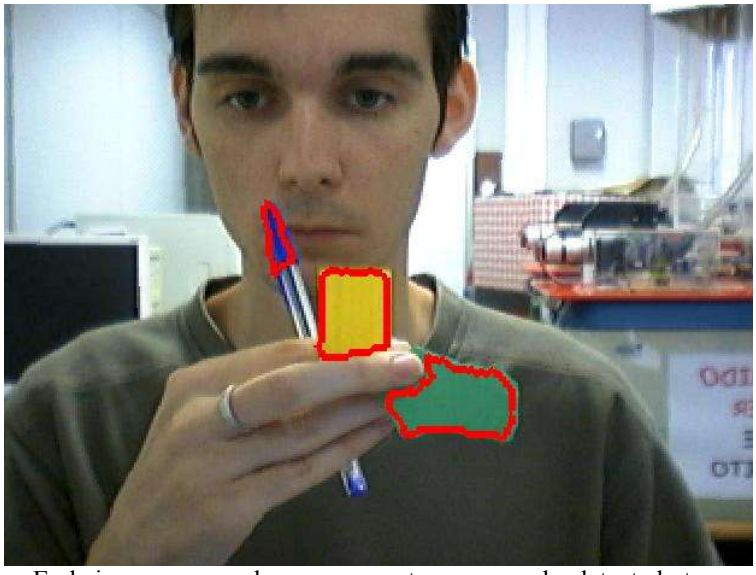


En la imagen, el programa ha detectado como patrón un trozo de cartulina verde, la pelota al colisionar contra ésta rebota.

Sample2: En este ejemplo se muestra como se pueden reconocer patrones de colores diferentes al mismo tiempo y detectar colisiones entre ellos.

Este ejemplo, al contrario de los demás, utiliza el fichero de configuración conf2.con que no puede editarse con el programa **Configuration**. Si quiere editarse debe realizarse manualmente ya que todavía no se ha implementado una utilidad que se encargue de ello.

Por defecto utiliza un color verde como dispositivo apuntador, y amarillo, azul y rojo para marcar los objetivos con los que puede interactuar el puntero. Cuando el puntero colisiona con uno de estos objetivos, el programa emite un sonido (utilizando las librerias Allegro) diferente según cual sea el objetivo con el que colisione.



En la imagen se puede ver como este programa ha detectado tres patrones de distinto color.

El formato del fichero conf2.con es parecido al de conf.con, pero con la diferencia de que conf2.con puede guardar los rangos de varios patrones:

```
Umbral   <--  La tolerancia permitida, es el mismo para todos los patrones
#   <--  carácter separador de patrones '#'
minH1 maxH1 <-- Los valores mínimo y máximo de H, del primer patrón
minS1 maxS1  <-- Los valores mínimo y máximo de S, del primer patrón
minV1 maxV1 <-- Los valores mínimo y máximo de V, del primer patrón
#   <--  carácter separador de patrones '#'
minH2 maxH2 <-- Los valores mínimo y máximo de H, del segundo patrón
minS2 maxS2  <-- Los valores mínimo y máximo de S, del segundo patrón
minV2 maxV2 <-- Los valores mínimo y máximo de V, del segundo patrón
#   <--  carácter separador de patrones '#'
. . . <--- otros patrones
#   <--  carácter separador de patrones '#'
minHn maxHn <-- Los valores mínimo y máximo de H, del último patrón
minSn maxSn  <-- Los valores mínimo y máximo de S, del último patrón
minVn maxVn <-- Los valores mínimo y máximo de V, del último patrón
-   <-- carácter que señala el final del fichero
```

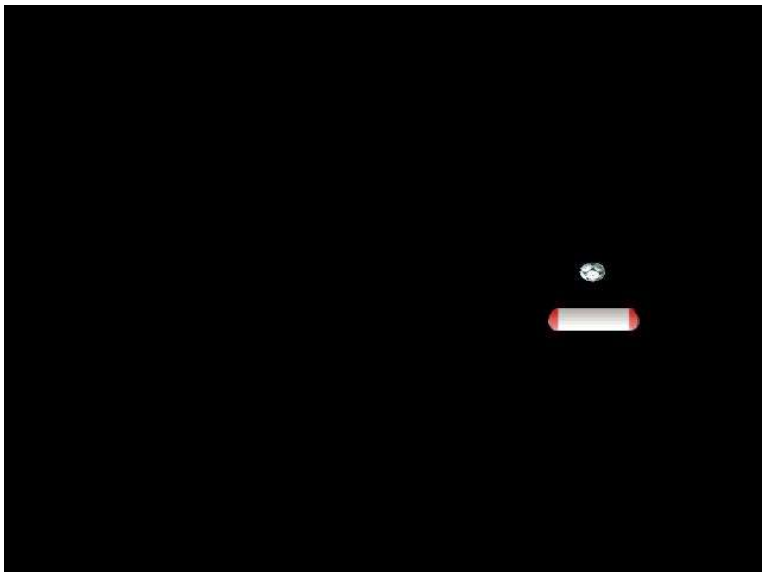
Hay que tener en cuenta que para el programa sample2 solo existen cuatro patrones. El que se encuentra primero en este fichero será aquel que haga de puntero, mientras que el resto serán los objetivos.

Si se desea modificar este fichero, se puede utilizar algún programa de edición de imágenes para obtener los valores HSV de un determinado color y añadirlos manualmente al fichero, o bien utilizar el programa Configuration, que aunque no está preparado para guardar información de varios patrones, puede servir para obtener uno por uno los valores HSV de cada patrón, y copiarlos luego en el fichero conf2.con.

Clicksample: Este ejemplo muestra como podemos imitar el comportamiento de un ratón mediante el patrón elegido. Para ello se ha redefinido el *click* del ratón, de forma que para hacer un

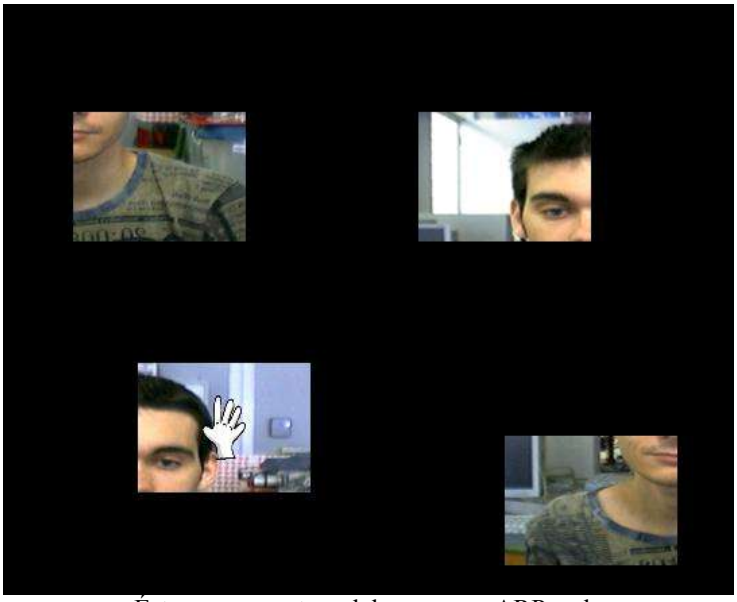
click debemos ocultar nuestro patrón durante un pequeño instante de tiempo y volverlo a mostrar. De esta forma podemos hacer *click* sobre una pelotita, arrastrarla a cualquier punto de la pantalla y soltarla mediante un nuevo *click*.

Pelotita: Se trata de una evolución del programa **Sample** al que se le ha eliminado la imagen de vídeo de la cámara y se ha sustituido en la imagen el patrón por una barra que representa al jugador. De esta manera se muestra como nuestro patrón puede también ser utilizado como un joystick sencillo en un juego. Además se han añadido algunas normas de juego, texturas y otros detalles con el fin de hacerlo más divertido y atractivo.



Esta imágen se ve como ha cambiado el programa sample, mediante imagenes y la eliminación de la imagen de la cámara.

Arpuzzle: Es una evolución del ejemplo **Clicksample** en la que el usuario podrá hacer algo más que arrastrar una pelotita por la pantalla. Se muestra una pantalla con 4 piezas separadas de una foto que el usuario tendrá que montar, para ello utilizará el patrón que haya elegido y haciendo *click* (ocultando y mostrando el patrón rápidamente) sobre alguna de ellas, podrá moverla por la pantalla y soltarla en la posición que le corresponda mediante un segundo *click*.



Ésta es una captura del programa ARPuzzle

4. Modificando ARToolKit

4.1 Estudiando el código

El primer paso para tratar de lograr nuestro objetivo fue analizar el código de los ejemplos y de las funciones que proporcionaban las librerias ARToolKit para comprender su funcionamiento y así poder encontrar algo que nos permitiera identificar patrones más naturales.

Para facilitar la tarea de comprender como se llevaba a cabo la identificación de los patrones, se realizó una especie de diagrama de estructura en el que se representaban aquellos ficheros, llamadas a funciones y datos intercambiados que tomaban parte en la identificación de los patrones.

El funcionamiento deducido del diagrama es básicamente el siguiente:

1. El programa de Realidad Aumentada utiliza la función arVideoGetImage para capturar una imagen de la cámara. Y la muestra con la función argDispImage.
2. Luego realiza una llamada a la función arDetectMarker, pasándole como parámetro la imagen capturada por la cámara, y esperando como respuesta el número de patrones detectados y la información correspondiente a cada uno de estos.
3. Por su parte, la función arDetectMarker realiza una llamada a otra función, llamada arLabeling, pasándole la imagen capturada y esperando recibir una imagen etiquetada con las zonas detectadas.
4. La funcion arLabeling invoca a la subfunción adecuada (labeling2 o labeling3) y los resultados que obtenga de ésta serán los que al final se devuelvan a la funcion arDetectMarker.
5. La subfuncion labeling (labeling2 o labeling3) se encarga de comparar cada punto de la imagen con un cierto umbral de intensidad, de forma que obtiene todas las zonas de la imagen que cumplen con cierta condición de intensidad, es decir, todas aquellas zonas que, a priori, tienen posibilidad de contener uno de los patrones.
6. Una vez obtenida esta información de la imagen, el control pasa de nuevo a la función arDetectMarker, la cual se encargará de utilizar la información obtenida para invocar a la función arDetectMarker2.
7. La función arDetectMarker2 básicamente se encarga de obtener la información de contorno de las zonas detectadas por arLabeling invocando para ello a arGetContour. Y de descartar aquellas zonas que no son cuadradas mediante la funcion check_square. También descarta aquellas zonas que son demasiado grandes o demasiado pequeñas, y finalmente devuelve el número de zonas encontradas que se consideran válidas.
8. Hasta este punto hemos obtenido todos aquellos posibles **cuadrados negros** que aparecen en la imagen y que no son demasiado grandes o pequeños. Posteriormente arDetectMarker comprobará si en su interior se encuentra alguna de las plantillas almacenadas y se realizarán operaciones para comprobar la orientación y posición de la plantilla, pero toda este proceso no nos interesa, así que no profundizaremos más en él.

Con esto hemos encontrado ya las funciones que parece que podrán sernos de ayuda: por una parte las funciones de **labeling** controlan las intensidades que se detectarán en la imagen. Por otra parte **Check_Square** comprueba la forma cuadrada de las marcas. Por tanto parece lógico que si queremos que nuestros programas detecten zonas de colores, en lugar de en escala de grises, y que no tengan porque ser cuadradas, tendremos que actuar de alguna manera sobre estas dos funciones.

4.2 Jugando con las restricciones

Ahora que conocemos las funciones clave, veamos como podemos modificarlasde forma que nos sean útiles.

En cuanto a la restricción de forma, una posibilidad es cambiar la funcion Check_Square por otra función que compruebe que el patrón tiene determinada forma, pero por su dificultad se ha obtado por dejar que el patrón tenga una forma libre, es decir, eliminar la restricción de forma, lo cual es fácil de conseguir simplemente eliminando la invocación a la función Check_Square. Comentar las siguientes líneas de la función arDetectMarker2 será suficiente:

```
// ret = check_square( warea[i], &(marker_info2[marker_num2]), factor );
// if( ret < 0 ) continue;
```

Con este cambio podemos detectar en la imagen ya no solo cuadrados negros, sino cualquier forma negra que aparezca en la imagen. La idea ahora, puesto que parece muy difícil añadir restricciones de forma a los patrones, es hacer algo con el color, conseguir de alguna manera que en lugar de comprobar la intensidad de cada píxel, compruebe sus valores RGB, de este modo podríamos buscar formas que no fueran negras o grises, sino del color que eligieramos.

La clave para realizar esto se encuentra en la función arLabeling (en realidad en las subfunciones labeling2 y labeling3), en las siguientes líneas:

```
#if defined(AR_PIX_FORMAT_ARGB)[1]
    if( *(pnt+1) + *(pnt+2) + *(pnt+3) <= thresh )
```

Donde pnt es un puntero a la posición del píxel de la imagen que está siendo analizado, y *(pnt+0), *(pnt+1), *(pnt+2) y *(pnt+3), son respectivamente los valores Alpha, Red, Green y Blue del píxel. Puede verse en este fragmento de código que las 3 componentes RGB se suman para obtener un valor de intensidad y compararlo con un umbral. Puesto que tenemos las 3 componentes RGB de la imagen, es fácil cambiar la condición por alguna de las siguientes:

```
if(*(pnt+1)>=120)  //el valor de la componente R es mayor o igual que 120
if(*(pnt+1)==255 && *(pnt+2)==0 && *(pnt+3)==0) //El valor del píxel es rojo puro
if(*(pnt+1)==255 && *(pnt+2)==255 && *(pnt+2)==0)//Amarillo puro
```

Sin embargo estas condiciones son tan restrictivas que difícilmente encontraremos algún objeto en el mundo real que las cumpla. Si queremos detectar objetos reales no podemos simplemente definir un color y ya está, debemos tener en cuenta que los distintos puntos del objeto pueden tener valores RGB diferentes según sea la iluminación y otras características del píxel. Por tanto nos vemos obligados a definir una cierta desviación del color elegido**.** Si queremos identificar, por ejemplo, un objeto tal que R=125, G=60 y B=7, la condición que usariamos sería algo así:

```
if((*(pnt+1)<125+desv && *(pnt+1)>125-desv)
  && (*(pnt+2)<60+desv && *(pnt+2)>60-desv)
    && (*(pnt+2)<7+desv && *(pnt+2)>7-desv))
```

1 Existe una comprobación de este tipo para cada modo de color soportado. De este modo se ejecutan las operaciones adecuadas según el modo de color en que está capturando la cámara. Hay que tener en cuenta que las componentes de color siguen el mismo orden con el que se identifica el color. Es decir, para el modo ARGB la primera componente (pnt+0) es A (Alpha), la segunda (pnt+1) R (Red), etc.

Donde *desv* es el valor de la desviación deseada, teniendo en cuenta que a valores más grandes de desviación le corresponden rangos menos restrictivos y por tanto mayor posibilidad de detectar la totalidad o mayoria de píxeles del objeto, pero también aumenta la posibilidad de error y la detección de píxeles no deseados. Además, el hecho de usar el espacio de color RGB aumenta la posibilidad de detectar zonas indeseadas usando este tipo de rangos, veremos que este problema puede solucionarse más adelante, en parte, mediante la utilización del espacio de color HSV.

4.3 Modificando las librerias

Tras realizar algunas pruebas modificando las funciones anteriores, es momento de dejarlo todo tal como estaba para evitar conflictos con el funcionamiento habitual de ARToolKit, queremos ampliarlo, no sustituirlo. Es por eso que a partir de ahora, para realizar las modificaciones oportunas, añadiremos nuevas funciones a las librerías en lugar de modificar las existentes, de este modo las librerías podrán seguir siendo usadas de la forma habitual a pesar de que las hayamos modificado.

Debemos tener en cuenta que estas funciones deben ser accedidas por un programa, y es por ello que debemos tener la precaución de definir la cabecera de estas funciones en el fichero include/AR.h, de lo contrario el programa no podrá acceder a las funciones.

Se han añadido las siguientes funciones al código fuente de las librerías (/lib/SRC/AR):

arDetectMarker3: Es una versión modificada de la función arDetectMarker2, a la que se le ha suprimido la invocacion a Check_Square, eliminando así la restricción de forma. Además se ha modificado de forma que devuelva correctamente la información y cantidad de marcas detectadas teniendo en cuenta que no tienen porque ser cuadradas, sin esta modificación se comportaría de forma extraña. Además hemos incluido su cabecera en el fichero AR.h para que pueda ser accedida desde fuera. Nuestra intención es poder invocar a esta función directamente desde un programa y saltarnos así la llamada a arDetectMarker. El código de esta función se encuentra en el fichero **arDetectMarker2.c**.

RGBtoHSV: Es una función que implementa un algoritmo para convertir los valores RGB obtenidos de la imagen a valores HSV, evitando así algunos de los inconvenientes de utilizar RGB, especialmente al utilizar rangos. Se encuentra en el fichero **arColor.c** que ha sido añadido al proyecto.

labelingHSV: Es una nueva función de etiquetado añadida al fichero **arLabeling.c**. Es parecida a labeling2 pero se han modificado las condiciones de detección de color de modo que la comparación de color se realiza usando el espacio HSV de la siguiente forma:

```
RGBtoHSV(R_COMP,G_COMP,B_COMP,&H,&S,&V);
if((((H>=minH && H<=maxH) && (S>=minS && S<=maxS)
   && (V>=minB && V<=maxB)) && !invert))
```

Primero se utiliza la función *RGBtoHSV* para obtener los valores HSV a partir de las componentes RGB del píxel. *R_COMP*, *G_COMP* y *B_COMP* son constantes que están definidas en el fichero **/include/formatopixel.h** y que son sustituidas por el preprocesador por **(pnt)*, **(pnt+1)*, **(pnt+2)* o **(pnt+3),* según sea el formato de color que se esté utilizando, para que apunten a las componentes RGB correspondientes del píxel.

Una vez obtenidas las componentes HSV, se comprueba que se encuentren dentro de los rangos decididos por el usuario para un patrón determinado. Estos rangos quedan determinados para cada componente por los valores maxH, minH, maxS, minS, minB y maxB que la función recibirá

como parámetros en su llamada.

labelingHSVMultiple: Otra función de etiquetado, definida también en el fichero arLabeling.c, que amplia la funcionalidad de labelingHSV permitiendo detectar varias zonas de color distintas, es decir, detecta varios patrones en la imagen. Esta vez se realizan las siguientes operaciones para determinar si se acepta o sea rechaza un píxel de la imagen:

```
detectado=0;
RGBtoHSV(R_COMP,G_COMP,B_COMP,&H,&S,&V);
   for(l=0;l<numHSV;l++) {
      if((H>=*(minH+l) && H<=*(maxH+l)) && (S>=*(minS+l)
       && S<=*(maxS+l)) && (V>=*(minB+l) && V<=*(maxB+l)))  {
                detectado=1;
                break;
               }
      }
if(detectado) {
...
 }
```

Esta vez tenemos un array de máximos y otro de mínimos valores aceptados para cada componente que la función recibe como parámetros. Tenemos que comprobar si cumple con al menos uno de los rangos, en tal caso el píxel se acepta. Para ello, una vez obtenidas las componentes HSV del píxel, se van recorriendo los arrays y se comprueba si los valores HSV entran dentro de cada rango. Si el píxel cumple con alguno de los rangos, el bucle termina y se marca como detectado, de lo contrario se sigue comprobando con el resto de rangos.

A parte de las nuevas funciones, también ha sido necesario modificar o crear otros ficheros:

Fichero include/AR/AR.h: Ha habido que añadirle las cabeceras de todas las nuevas funciones, ya que las ibamos a utilizar desde fuera del fichero en que fueron definidas.

Fichero include/formatopixel.h: Se ha creado para trabajar más fácilmente con las componentes RGB de un píxel en las funciones *labelingHSV* y *labelingHSVMultiple*. En el se encuentran definidas las constantes *R_COMP*, *G_COMP*, *B_COMP* y *A_COMP* teniendo en cuenta el modo de color utilizado, de esta manera ya no tenemos que comprobar el modo de color y podemos trabajar más cómodamente. También se define el número de componentes que tiene el píxel. Por ejemplo para el modo ARGB se tiene lo siguiente:

```
#if defined(AR_PIX_FORMAT_ARGB)
#define B_COMP *(pnt+3)
#define G_COMP *(pnt+2)
#define R_COMP *(pnt+1)
#define A_COMP *(pnt+0)
#define VALORES_POR_PIXEL 4
```

Donde se puede ver que cada componente almacena la posición en que se encuentra dentro del píxel, siendo *pnt* un puntero que guarda la dirección del píxel. Puesto que el modo es ARGB, primero se encuentra la componente A (posición 0), luego la R (posición 1), luego la G (posición 2) y finalmente la B (posición 3). Además vemos que existen 4 valores (componentes) por píxel: A, R, G y B. De forma parecida se han definido estas constantes para el resto de modos de color soportados.

5. Desarrollo de ejemplos y utilidades

5.1 Utilizando las nuevas funciones

Ahora que tenemos todas las funciones que necesitamos ya podemos comenzar a crear programas de ejemplo y utilidades que utilicen la información de los patrones (colores) detectados. Pero primero vamos a ver como deben usarse estas funciones, y que información proporcionan a nuestro programa. Veamos pues la estructura básica de un programa que utiliza nuestra ampliación de las librerías:

1. Inicializar el dispositivo de video y otros parámetros de ARToolKit.
2. Comenzar la captura de video. Para ello se utiliza la función: *arVideoCapStart()*;
3. Comenzar el bucle principal que se encargará de ir capturando imágenes, actualizando la información y gestionar eventos de teclado y ratón en cada iteración. Se utiliza la orden *argMainLoop(mouseEvent, keyEvent, mainLoop)*, donde *mainLoop* es el nombre que le hemos dado a la función que implementa el bucle principal y *mouseEvent* y *keyEvent* son, respectivamente, las funciones que manejan los eventos de ratón y teclado.
4. Se captura una imagen y se almacena su dirección de memoria en una variable puntero de tipo *ARUint8*:

```
dataPtr = (ARUint8 *)arVideoGetImage());
```

5. La imagen puede mostrarse en pantalla si se desea. Para ello le pasamos a *argDispImage* el puntero que apunta a la imagen y las coordenadas *x* e *y* de la pantalla donde se mostrará ésta: *argDispImage (dataPtr, 0, 0)*. Pero antes de eso debe usarse la función *argDrawMode2D()* que ajustará algunos parámetros para poder dibujar la imagen en dos dimesiones.
6. Utilizar nuestra función *arLabelingHSV* (o *arLabelingHSVmultiple*) para encontrar las zonas de color deseadas en la imagen. Como resultado obtendremos una imagen filtrada que almacenaremos en una variable puntero de tipo *ARint16*.

```
limage =  arLabelingHSB( dataPtr, thresh, &label_num, &area, &pos,
    &clip, &label_ref, 1, minH, maxH, minS, maxS, minB, maxB, invert)
```

   Donde *dataPtr* es la imagen capturada; thresh un valor de umbral que actualmente no sirve para nada; *&label_num* es una variable que contendrá el número de zonas detectadas en la imagen tras la invocación de la función; *&area* es un array que contendrá las áreas (número de píxels) de cada una de las zonas detectadas; *&pos* es un array que contendrá la posición central de cada zona detectada; *&clip* es un array que para cada zona detectada define la zona rectangular más pequeña que la contiene; *&label_ref* es un array con algunos parámetros sobre las zonas detectadas que utilizan mas adelante otras funciones; 1 corresponde a un parámetro interno de la funcion que para nosotros siempre será 1; *minH, maxH, minS, maxS, minB* y *maxB* son los mínimos y máximos que definen los rangos de color HSV que queremos detectar; y *invert* es un valor booleano que indica si se deben buscar los valores dentro o fuera de los rangos proporcionados (actualmente poco perfeccionado) y que por lo general tendrá como valor 0.

7. Invocar a nuestra función *arDetectMarker3* a través de la que obtendremos un array con todas los patrones detectados. Cada uno de estos patrones será una estructura de tipo *ARMarkerInfo2* (definida en **/include/AR/AR.h** en las librerías originales). La invocación se realizará así:

```
marker2 = arDetectMarker3( limage, label_num, label_ref, area, pos, clip,
            AR_AREA_MAX, AR_AREA_MIN, &marker_num2);
```

Donde *marker2* es una variable de tipo *ARMarker2; limage* es la imagen tratada por *arLabelingHSB*; *label_num, label_ref, area, pos* y *clip* son las mismas variables que se pasaron a *arLabelingHSB*; *AR_AREA_MAX* y *AR_AREA_MIN* son constantes que almacenan los valores máximo y mínimo aceptados para el área de los patrones; y *&marker_num2* es una variable que se pasa como puntero y que contendrá el número de patrones (estructuras de tipo *ARMarkerInfo2*) existentes en la imagen tras la ejecución de esta función.

8. Ahora que hemos localicado los patrones de la escena, disponemos de una estructura de tipo *ARMarkerInfo2* para cada uno de éstos. Esta estructura almacena la siguiente información:

   area: es el número de píxeles que contiene.
   pos[]: Almacena la posición en pantalla del centro de la imagen. Es un array que en su posición 0 almacena la coordenada *x* y en la 1 la coordenada *y*.
   coord_num: Contiene el número de puntos que forman el entorno del patrón.
   x_coord[]: Es un array que contiene la coordenada *x* de cada píxel del contorno.
   y_coord[]: Es un array que contiene la coordenada *y* de cada píxel del contorno.

   Las operaciones que se realicen a partir de aquí dependerán del tipo de programa que se realice, pero consistirán en utilizar de algún modo la información obtenida de los patrones detectados.

5.2 El primer programa: la utilidad de configuración.

En principio este programa servía para comprobar el correcto funcionamiento de las modificaciones realizadas, mediante el marcado del contorno de los patrones detectados. Posteriormente ha evolucionado para convertirse en la utilidad de configuración cuya descripción y utilización se vieron con anterioridad. A continuación se explica detalladamente como se ha creado.

Pensemos un poco que requisitos deberá cumplir este programa:

a. Debe permitir capturar y mos trar la imagen de la cámara
b. Debe permitirnos detectar patrones de un determinado color
c. Debe ofrecer cierta información sobre los patrones detectados

Hasta aquí se sigue la estructura básica explicada en el punto anterior, pero además:

d. Debe permitir al usuario seleccionar de forma fácil el color de su patrón mediante ratón o teclado.
e. Debe marcar el contorno de los patrones detectados, con el fin de que el usuario sepa en cada momento que zonas de la escena están siendo detectadas.
f. Debe permitir guardar la información obtenida en la sesión en un fichero de configuración (conf.con) y cargar la información de dicho fichero al inicio de cada nueva sesión.

Veamos como realiza todo esto el programa:

Lo primero es incluir los ficheros de cabecera con cabeceras de funciones y/o constantes que vamos a necesitar. Podemos ver que estos pueden variar de un sistema operativo a otro, por ello se incluyen las sentencias *#ifdef* y *#ifndef*, que permiten conocer en que sistema operativo se realiza la compilación del programa. Por el momento solo están soportados los sistemas Windows y Linux.

```
#ifdef _WIN32
#  include <windows.h>
#endif
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>


#ifndef __APPLE__
#  include <GL/glut.h>
#else
#  include <GLUT/glut.h>
#endif
#include <AR/gsub.h>
#include <AR/ar.h>
#include <AR/video.h>
#include <formatopixel.h>
```

Definimos algunas constantes que necesitaremos mas tarde. Concretamente creamos una constante para cada mínimo y máximo valor de HSV que contendrán los valores por defecto que detectará el programa hasta que el usuario seleccione otros, en nuestro caso éstos valdrán 0 por defecto. Tenemos también un umbral, incialmente con valor 5, que es una desviación aplicada sobre los máximos y mínimos de cada componente.

```
#define MIN_H 0
#define MAX_H 0
#define MIN_S 0
#define MAX_S 0
#define MIN_B 0
#define MAX_B 0
#define UMBRAL 5
```

Declaramos algunas variables que necesitaremos mas tarde: *xsize* e *ysize* contendrán más adelante la anchura y la altura de la ventana en que se ejecutará la aplicación; *thresh* es un valor de umbral que no utilizamos nada más que para pasarselo a *arLabelingHSB*; *minH, maxH, minS, maxS, minB* y *maxB* son las variables que contienen los valores de los mínimos y máximos valores de cada componente HSV que se buscarán en la imagen; *umbral* es la variable que contiene la desviación; *invert* es una variable booleana que sirve para decidir si el color elegido se busca dentro de los rangos seleccionados (*invert = 0*) o fuera (*invert=1*); y *dataPtr* contendrá un puntero a la imagen.

```
int           xsize, ysize;
int           thresh = 100;
int           minH = MIN_H, maxH = MAX_H, minS = MIN_S, maxS = MAX_S,
              minB = MIN_B, maxB = MAX_B, umbral = UMBRAL;
int invert = 0;
ARUint8       *dataPtr;
```

Si nos encontramos en Windows, podemos establecer los parámetros de configuración de la cámara escribiendolos en una variable de tipo string *(char*)* llamada *vconf*, le pasaremos el parámetro *flipV* que sirve para invertir la imagen en vertical, de lo contrario se vería al revés. Le pasaremos también el parámetro *showDlg* que sirve para que antes de comenzar la ejecución del programa, se nos presente una ventana para configurar algunos parámetros de la cámara. Si nos encontramos en Linux creamos la variable pero dejamos la configuración para más tarde.

```
#ifdef _WIN32
char                    *vconf = "flipV,showDlg"; // see video.h for a list of supported
parameters
#else
char                    vconf[256];
#endif
```

Las siguientes variables sirven para configurar la cámara a partir de un fichero, ya existente, que contiene los parámetros que se utilizarán por defecto.

```
char        *cparam_name    = "Data/camera_para.dat";
ARParam     cparam;
```

Declaramos las cabeceras de las funciones que utilizaremos en el programa: una función *init* para inicializar algunos parámetros y variables del programa; una función *cleanup* que libera los recursos utilizados; una función *keyEvent* para gestionar los eventos de teclado; una función *mouseEvent* para manejar eventos del ratón; un función *mainLoop* que implementa el bucle principal de la aplicación que se repetirá hasta que el programa finalice; una función *guardarEnFichero* para guardar en un fichero la configuración; y una función *cargarDeFichero* para cargar la configuración guardada.

```
static void   init(void);
static void   cleanup(void);
static void   keyEvent( unsigned char key, int x, int y);
static void   mouseEvent(int button, int state, int x, int y);
static void   mainLoop(void);
static int    guardarEnFichero(void);
static int    cargarDeFichero(void);
```

La función init realiza varias tareas de inicialización: se encarga de *cargarDeFichero* la configuración guardada; abre el dispositivo de vídeo utilizando la configuración almacenada en *vconf*; obtiene el ancho y largo de la ventana y los guarda en las variables *xsize* y *ysize*; carga los parámetros iniciales de la cámara y crea la ventana del programa:

```
static void init( void )
{
   ARParam  wparam;
   cargarDeFichero();
   /* open the video path */
   if( arVideoOpen( vconf ) < 0 ) exit(0);
   /* find the size of the window */
   if( arVideoInqSize(&xsize, &ysize) < 0 ) exit(0);
   printf("Image size (x,y) = (%d,%d)\n", xsize, ysize);

   /* set the initial camera parameters */
   if( arParamLoad(cparam_name, 1, &wparam) < 0 ) {
      printf("Camera parameter load error !!\n");
      exit(0);
   }
   arParamChangeSize( &wparam, xsize, ysize, &cparam );
   arInitCparam( &cparam );
   printf("*** Camera Parameter ***\n");
   arParamDisp( &cparam );
   /* open the graphics window */
   argInit( &cparam, 2.0, 0, 0, 0, 0 );
}
```

La función cleanup detiene la captura, cierra el dispositivo de vídeo y destruye la ventana.

```
static void cleanup(void)
{
   arVideoCapStop();
   arVideoClose();
   argCleanup();
}
```

La función main puede recibir parámetros tal como vemos en su cabecera. Si la ejecución se realiza en Linux, podemos pasarle como parámetro la ruta del dispositivo de vídeo (por ejemplo */dev/video* o la ruta de un enlace al dispositivo de vídeo):

```
int main(int argc, char** args) {
```

Ahora es el momento de rellenar, en Linux, la variable *vconf* con los parámetros de configuración tal y como hicimos con anterioridad en Windows, lo haremos de forma que el usuario pueda elegir el dispositivo de vídeo a utilizar. Para ello primero copiamos a *vconf* los parámetros por defecto. Se asume por defecto que existe un enlace al dispositivo de video con el nombre "camara".

```
#ifndef _WIN32
char dev[256];
strcpy(vconf,"-dev=camara -channel=0 -palette=YUV420P -width=320 -height=240");
```

Si se han pasado por parámetro el dispositivo de video, primero se copian a la variable *vconf* los parámetros por defecto (excepto el parámetro -dev que establece el dispositivo de video) y luego se añade (se concatena) a la variable el parámetro recibido y que contiene la ruta al dispositivo de vídeo:

```
if(argc==2) {
 strcpy(vconf,"-channel=0 -palette=YUV420P -width=320 -height=240 -dev=");
 strcat(vconf, args[1]);
}
```

Si no se envió la ruta al dispositivo de vídeo como parámetro, se utilizarán los parámetros por defecto, pero primero comprobaremos si existe el enlace "camara" al dispositivo de vídeo en el directorio de ejecución, cosa que hacemos intentando abrir dicho fichero en modo lectura. Si el enlace no existe se ofrece la posibilidad de escribir la ruta del dispositivo o el enlace correspondiente:

```
else {
   FILE *f;
   f=fopen("camara","r");
   if(f == NULL) {
    printf("No se ha encontrado un enlace \"./camara\" al dispositivo de video.    Por favor,
indique el nombre de dispositivo (normalmente /dev/video0 o    /   dev/video1) en el que se
encuentra su camara: ");
    scanf("%s",&dev);
    strcpy(vconf, "-channel=0 -palette=YUV420P -width=320 -height=240                -
dev=");
    strcat(vconf, dev);
 }
   else {
   fclose(f);
 }
```

```
}
#endif
```

Después de esto, tanto si estamos en Windows como en Linux, mostramos por consola información sobre el modo de usar el programa:

```
printf("\nEste programa sirve para entrenar al sistema a reconocer el objetivo a  seguir:");
printf("\n\n\n\t1.Situe el objetivo delante de la camara.");
printf("\n\n\t2.Cuando vea su objetivo en la pantalla, haga click sobre el\n\tcon el boton derecho
 del raton.");
printf("\n\n\t3.Realice varios clicks sobre la superficie del objetivo a seguir\n\tcon el boton
 izquierdo del raton, hasta que la superficie de este\n\taparezca  totalmente bordeada.");
printf("\n\n\t4.Tambiï¿½ puede ajustar manualmente los rangos de H, S y V que\n\tse
detectaran,pulsando respectivamente las teclas 'h', 's' y 'b'.");
printf("\n\n\t5.Puede ajustar un umbral pulsando 'u'. Cuanto mayor sea el umbral,\n\tmayor sera la
 desviacion permitida respecto de las componentes\n\tHSV elegidas.");
printf("\n\n\t6.Cuando termine pulse 'g' para guardar la configuracion en el\n\tfichero
 conf.con.\n\n\n");
```

Terminamos la función *main* realizando una llamada a la función *init* (que ya hemos visto anteriormente), a *arVideoCapStart,* que inicia la captura de vídeo y a *argMainLoop*, que inicia el bucle principal.

```
  init();
  //start video capture
  arVideoCapStart();
  //start the main event loop
  argMainLoop( mouseEvent, keyEvent, mainLoop );
  return 0;
}
```

La función mainLoop implementa el bucle principal de la aplicación, y es por tanto la encargada de ir capturando imágenes cada cierto tiempo y actualzanzo la información del programa.

```
static void mainLoop(void) {
```

En esta función tendrá lugar la mayoría del proceso de detección de patrones y utilización de la información obtenida de éstos. Comenzamos declarando las variables que necesitaremos más adelante: *marker2* es un array de estructuras de tipo *ARMarkerInfo2*, es decir, un array de patrones detectados; *marker_num2* es el número de patrones detectados en la imagen; *i* y *j* son variables utilizadas como contadores en bucles; *limage* apuntará a la imagen tratada por *arLabelingHSB*; *label_num*, *area, clip*, *label_ref* y *pos* son parámetros que hay que pasar a las funciones *arLabelingHSB* y *arDetectMarker3* y que ya se explicaron anteriormente:

```
ARMarkerInfo2   *marker2;
int             marker_num2;
int             i,j;
ARInt16     *limage;
 int             label_num;
 int             *area, *clip, *label_ref;
 double          *pos;
 marker_num2 = 0;
```

Después de esto capturamos una imagen a partir del dispositivo de vídeo, siendo *dataPtr* un puntero a la imagen obtenida. Si la función *arVideoGetImage* devuelve *NULL*, no se puede

capturar imagen y el programa finaliza.

```
if( (dataPtr = (ARUint8 *)arVideoGetImage()) == NULL ) {
   arUtilSleep(2);
   return;
}
```

Mostramos la imagen en la ventana del programa, cumpliendo así con el requisito *a* del programa:

```
argDrawMode2D();
argDispImage( dataPtr, 0,0 );
```

Establecemos un color y grosor de línea para pintar el contorno de los patrones detectados, utilizaremos el color rojo:

```
glColor3f( 1.0, 0.0, 0.0 );
glLineWidth(6.0);
```

Se utiliza *arLabelingHSB* para detectar el color deseado en la imagen capturada (*dataPtr*) obteniendose una imagen filtrada (*limage*). Si el resultado de la función es 0, el programa finaliza. Con esto cumplimos con el requisito *b*.

```
limage = arLabelingHSB( dataPtr, thresh,
        &label_num, &area, &pos, &clip, &label_ref, 1,
                minH, maxH, minS, maxS, minB, maxB, invert);
if( limage == 0 )   {
  cleanup();
  exit(0);
}
```

Para cumplir el requisito *c,* la imagen obtenida por *arLabelingHSB* se pasa a *arDetectMarker3,* quien se encargará de obtener los píxeles que forman su contorno y de obtener otra información, devolviendo un array de patrones detectados. Si la función devuelve 0 el programa finaliza.

```
marker2 = arDetectMarker3( limage, label_num, label_ref,area, pos,      clip,
AR_AREA_MAX, AR_AREA_MIN, marker_num2);
 if( marker2 == 0 ) {
     cleanup();
      exit(0);
 }
```

Para finalizar, la función *mainLoop*, para cada patrón detectado (el número de patrones se encuentra en *marker_num2*), para cada punto de su contorno se dibuja una recta (utilizando la funcion *argLineSeg* de ARToolKit) desde el punto actual hasta el siguiente. De este modo dibujamos todo el contorno del patrón, cumpliendo así con el requisito *e*.

```
 for( i = 0; i < marker_num2; i++ ) {
     for(j=0; j<marker2[i].coord_num-1; j++) {
        argLineSeg( marker2[i].x_coord[j] ,  marker2[i].y_coord[j],
        marker2[i].x_coord[j+1],  marker2[i].y_coord[j+1], 0, 0);
     }
 }
 /*swap the graphics buffers*/
 argSwapBuffers();
}
```

La función keyEvent, es la encargada de manejar los eventos de teclado. Será invocada cada vez que el usuario presione una tecla, recibiendo como parámetros el código de la tecla presionada, y las coordenadas *x* e *y* del cursor del ratón.

```
static void  keyEvent( unsigned char key, int x, int y)
```

Pensemos cuales son las teclas que el programa debe detectar y cuales serán las acciones a realizar cuando se presione una de estas teclas:

Tecla Escape (código 0x1b): se liberan los recursos utilizados y el programa finaliza.

```
if( key == 0x1b ) {
   cleanup();
   exit(0);
}
```

Tecla 'h': muestra los valores actuales de *minH* y *maxH* (valores mínimo y máximo de la componente H, tono, que se detectarán en la imagen) y permite modificarlos (requisito d).

```
if( key == 'h' ) {
        printf("\nminH = %d, maxH = %d\n",minH,maxH);
   printf("\nIntroduce el valor minimo de H(tono):\n");
        scanf("%d",&minH);
   printf("\nIntroduce el valor maximo de H(tono):\n");
        scanf("%d",&maxH);

}
```

Tecla 's': muestra los valores actuales de *minS* y *maxS* (valores mínimo y máximo de la componente S, saturación, que se detectarán en la imagen) y permite modificarlos (requisito d).

```
if( key == 's' ) {
                printf("\nminS = %d, maxS = %d\n",minS,maxS);
               printf("\nIntroduce el valor minimo de S(Saturacion):\n");
                scanf("%d",&minS);
               printf("\nIntroduce el valor maximo de S(Saturacion):\n");
                scanf("%d",&maxS);
}
```

Tecla 'b': muestra los valores actuales de minB y maxB (valores mínimo y máximo de la componente V de HSV, brillo, que se detectarán en la imagen) y permite modificarlos (requisito d).

```
   if( key == 'b' ) {
                    printf("\nminB = %d, maxB = %d\n",minB,maxB);
                    printf("\nIntroduce el valor minimo de B(brillo):\n");
                     scanf("%d",&minB);
                    printf("\nIntroduce el valor maximo de B(brillo):\n");
                     scanf("%d",&maxB);
   }
```

Tecla 'i': esta tecla activa y desactiva la variable *invert.* Cuando esta variable esta activa, la función labeling buscará los colores fuera del rango elegido en lugar de dentro. En principio puede servir para detectar objetos (de cualquier color, excepto el elegido) que tienen de fondo el color elegido, pero no funciona correctamente.

```
if( key == 'i' ) {

    if(invert) invert = 0;
     else invert = 1;
   printf("\nInversion realizada\n");

}
```

Tecla 'u': muestra la desviación actual y permite modificarla.

```
if( key == 'u' ) {

   printf("\nIntroduce el umbral:\n");
    scanf("%d",&umbral);

}
```

Tecla 'g': muestra los valores actuales de los rangos seleccionados y permite guardarlos al fichero conf.con (requisito *f*).

```
if( key == 'g' ) {

 printf("\nValores actuales: \nH: [%d-%d]\nS: [%d-%d]\nB: [%d-%d]\n",
     minH,maxH,minS,maxS,minB,maxB);
 printf("\nDesea guardar los datos de configuracion de la sesion actual?(s/n)\n");
 scanf("%c",&resp);
 if(resp=='s') {
     if(guardarEnFichero()) printf("\nConfiguracion guardada en fichero.\n");
      else {printf("\nError!. No se pudo guardar la configuracion en el
     fichero.\n");}
     }
 }
```

La función mouseEvent, es la función manejadora del ratón, básicamente su objetivo es cumplir con el requisito *d*. Esta función, recibe como parámetros el botón del ratón sobre el que se realiza la acción, su estado y las coordenadas *x* e *y* del cursor.

```
static void mouseEvent(int button, int state, int x, int y)
```

Declaramos algunas variables. Las coordenadas *x* e *y* deben dividirse por dos para poder calcular más tarde la posición del píxel seleccionado.

```
double H, S, V;
ARUint8 *pnt;
x=x/2;
y=y/2;
```

Al hacer *click* (*state = GLUT_DOWN*) con el botón derecho del ratón (*button = GLUT_RIGHT_BUTTON*), se inicializan todos los máximos y mínimos al valor del píxel seleccionado, sumándo la desviación a los máximos y restándola a los mínimos.

```
if (button == GLUT_RIGHT_BUTTON && state == GLUT_DOWN) {
```

La variable *pn*t es un puntero de tipo *ARuint8* (un puntero a un byte), tenemos que hacer que apunte al primer byte correspondiente al píxel seleccionado. Para ello utilizamos la siguiente fórmula:

> Posición del byte = (Dirección del primer byte de la imagen + (Pixeles por fila * Numero de bytes de cada píxel * (coordenadaY -1)) + (CoordenadaX * Número de bytes de cada píxel))

Que aplicada a nuestro problema queda:

```
pnt = (dataPtr+(arImXsize*VALORES_POR_PIXEL*(y-1))+(x*VALORES_POR_PIXEL));
```

Donde *arImXsize* es el tamaño en X de la imagen (número de píxeles por fila), *VALORES_POR_PIXEL* se encuentra declarada en **/include/formatoPixel.h** y es el número de bytes de cada píxel según el modo de color utilizado y *dataPtr* es el puntero a la imagen (dirección al primer byte de la imagen).

Una vez obtenida la posición del píxel y asignada a la variable *pnt*, utilizamos la función *RGBtoHSV* para transformar los valores RGB del píxel al espacio HSV. Recordemos que *R_COMP*, *G_COMP* y *B_COMP* estaban definidas en **/include/formatoPixel.h** a partir de la variable *pnt* de manera que, en cualquier momento, cada una de ellas contiene el valor de la componente R, G o B del píxel al que apunta la variable *pnt*.

```
RGBtoHSV(R_COMP, G_COMP, B_COMP, &H, &S, &V);
```

A continuación se asigna a cada máximo el valor (H, S o V) del píxel seleccionado sumándole el umbral o desviación. Lo mismo sucede con los mínimos con la excepción de que la desviación se resta.

```
    minH=H-umbral;
     maxH=H+umbral;
     minS=S-umbral;
     maxS=S+umbral;
     minB=S-umbral;
     maxB=S+umbral;
     printf("\nValores del pixel:\n%f %f %f\n",H, S, V);

}
```

En caso de que el click (*state = GLUT_DOWN*) se realice con el botón izquierdo del ratón (*button = GLUT_LEFT_BUTTON*), se actualizan todos los máximos y mínimos para que el rango incluido incluya al píxel seleccionado. Es decir, se amplia el rango (aumenta el máximo o disminuye el mínimo) de modo que, además de los píxeles que ya estaban incluidos dentro del rango, se incluya también al nuevo píxel.

```
if (button == GLUT_LEFT_BUTTON && state == GLUT_DOWN) {
```

La posición del primer byte del píxel se asigna a pnt de la misma manera que lo hicimos anteriormente, y se invoca a la función *RGBtoHSV* para transformarlo de RGB a HSV.

```
pnt = (dataPtr+(arImXsize*VALORES_POR_PIXEL*(y-1))+(x*VALORES_POR_PIXEL));
RGBtoHSV(R_COMP, G_COMP, B_COMP, &H, &S, &V);
```

Posteriormente, se comprueba si el valor de alguna de las componentes del píxel seleccionado es mayor que el actual máximo (menos el umbral), en cuyo caso el nuevo máximo será el de la

componente del píxel seleccionado (más el umbral). Se comprueba también para cada componente, si es menor que el actual mínimo (más el umbral), en este caso hay que actualizar el mínimo asignándole el valor de la componente del píxel seleccionado (menos el umbral). De este modo conseguimos que el píxel seleccionado siempre quede dentro del rango.

```
    if(H>=(maxH-umbral)) {maxH=(H+umbral);}
    if(H<=(minH+umbral)) {minH=(H-umbral);}
          if(S>=(maxS-umbral)) {maxS=(S+umbral);}
    if(S<=(minS+umbral)) {minS=(S+umbral);}
    if(V>=(maxB-umbral)) {maxB=(V+umbral);}
    if(V<=(minB+umbral)) {minB=(V-umbral);}

    printf("\nValores del pixel:\n%f %f %f\n",H, S, V);
    printf("\nNuevos rangos: \nH: [%d-%d]\nS: [%d-%d]\nB: [%d-%d]
        \n",minH,maxH,minS,maxS,minB,maxB);

}
```

De esta forma podemos detectar un patrón en la imagen fácilmente, si primero hacemos *click* sobre uno de sus píxeles con el botón derecho, y posteriormente vamos haciendo clicks en las zonas no detectadas del objeto para ampliar los rangos y detectar la mayor parte de los píxeles del patrón.

La función guardarEnFichero, como su propio nombre indica, permite guardar la configuración actual (máximos, mínimos y desviación) al fichero conf.con (requisito *f*). Su funcionamiento es muy simple, simplemente se abre el fichero para escritura y se escriben los datos siguiendo la estructura del fichero conf.con que se explicó con anterioridad.

```
static int guardarEnFichero(void) {
  FILE *fich;
  fich=fopen("conf.con","w");
  if(fich != NULL) {
  fprintf(fich,"%d\n#\n%d %d\n%d %d\n%d %d\n-
\n",umbral,minH,maxH,minS,maxS,minB,maxB);
  fclose(fich);
  return 1;
 }
 return 0;
}
```

La función cargarDeFichero, es la encargada de leer el fichero conf.con y obtener de él la información sobre los valores guardados para asignarselos a las variables del programa (requisito *f*). Se abre el fichero conf.con en modo lectura y se lee cada valor, siguiendo la estructura propia del fichero, asignándoselo a la variable correspondiente.

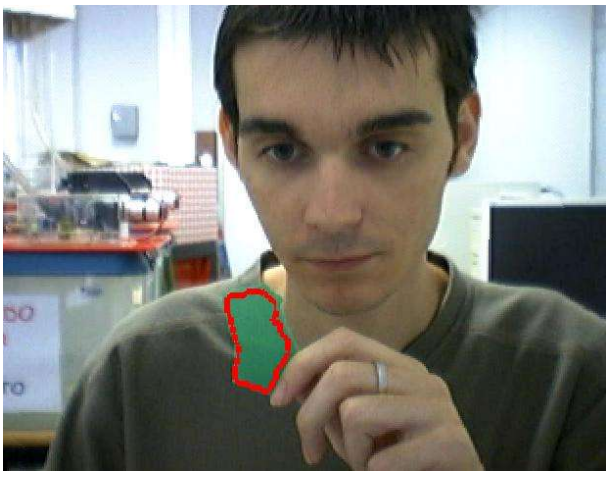
```
static int cargarDeFichero(void) {
 FILE *fich;
 char c;
 fich=fopen("conf.con","r");

 if(fich != NULL) {
     fscanf(fich,"%d",&umbral);
     fscanf(fich," %c",&c);

    if(c=='#'){  //si el separador es correcto y no es fin de fichero carga valores;
    fscanf(fich,"%d",&minH);
    fscanf(fich,"%d",&maxH);
```

```
    fscanf(fich,"%d",&minS);
    fscanf(fich,"%d",&maxS);
    fscanf(fich,"%d",&minB);
    fscanf(fich,"%d",&maxB);
   }
 fclose(fich);
 return 1;
}
return 0;
}
```

Con esto ya tenemos una herramienta que, además de permitirnos comprobar el correcto funcionamiento de las librerias, nos permitirá guardar un fichero de configuración con los valores que el resto de programas utilizarán para detectar al patrón.



5.3 Interacción con objetos del mundo real. Detectando varios patrones

Existe la posibilidad, usando las nuevas funciones agregadas a las librerías, de detectar varios patrones (colores) al mismo tiempo en la imagen. Esto nos permite, por ejemplo, detectar interacciones entre objetos del mundo real. El programa **Sample2** es un ejemplo de como podemos crear un programa que detecte la interacción entre un patrón puntero de un color, y varios patrones objetivo de otros colores, y realizar una acción diferente para cada posible interacción. En este programa se ha optado por emitir un sonido diferente según sea el color del patrón con el que interactúa (colisiona) el puntero, pero la acción a realizar podría haber sido cualquier otra. Veamos con detalle como funciona este programa[1]:

Primero se incluyen los ficheros de cabecera necesarios. Hay que tener en cuenta que para generar sonido se han utilizado las librerías Allegro, es por ello necesario incluir el fichero *allegro.h*, además en el caso de Windows, es necesario incluir el fichero *winalleg.h* en sustitución del *windows.h*.

```
#include <allegro.h>
#ifdef _WIN32
#  include <winalleg.h>
#endif
#include <stdio.h>
#include <stdlib.h>
```

1 Algunas funciones y variables de este programa coinciden con las del programa **Configuration** que se ha explicado anteriormente, y por tanto no se explicarán.

```
#include <string.h>
#include <math.h>

#ifndef __APPLE__
#  include <GL/glut.h>
#else
#  include <GLUT/glut.h>
#endif
#include <AR/gsub.h>
#include <AR/param.h>
#include <AR/ar.h>
#include <AR/video.h>
#include <time.h>
```

A continuación definimos un umbral, y creamos algunas variables que necesitaremos más tarde para reproducir el sonido.

```
#define UMBRAL 5
AUDIOSTREAM *stream;
unsigned char *p;
int offset=0;
int        sonando=0;
```

Creamos un array para cada mínimo y para cada máximo de cada componente HSV. Estos arrays contendrán los diferentes rangos de colores que se deben detectar, teniendo en cuenta que el del objeto puntero se encontrará en la posición 0 de los array.

```
int  minH[10], maxH[10], minS[10], maxS[10], minB[10], maxB[10],
     numHSV=0, umbral = UMBRAL;
```

Luego declaramos las cabeceras de las funciones del programa: Las funciones *init* y *cleanup* son funciones de inicialización y de destrucción respectivamente. *KeyEvent* es la función manejadora de eventos de teclado. La función *mailLoop* implementa el bucle principal del programa. Las funciones *guardarEnFichero* y *cargarDeFichero* permiten guardar y cargar en fichero varios patrones. Y la función *generate_audio* genera un sonido dependiendo del tono recibido como parámetro.

```
static void   cleanup(void);
static void   init(void);
static void   keyEvent( unsigned char key, int x, int y);
static int    generate_audio(unsigned char tono);
static void   mainLoop(void);
static int    guardarEnFichero(void);
static int    cargarDeFichero(void);
```

La función cleanup es idéntica a la vista en el programa **Configuration**, así que no volverá a explicarse.

La función init, es parecida a la del programa configuration, el único cambio reside en la sigiente línea:

```
if(!cargarDeFichero()) { minH[0]=0;maxH[0]=0;minS[0]=0;maxS[0]=0;
minB[0]=0;maxB[0]=0;}
```

Esta línea sirve para inicializar a 0 todos los mínimos y máximos del patrón puntero en caso de

que no se haya podido cargar la configuración del fichero.

La función keyEvent, es practicamente la misma que la del programa **Configuration**, así que no consideramos necesario volver a explicarla.

La función main, es muy parecida a la de **Configuration**, con la diferencia de que no se ha incluido un manejador de ratón (*mouseEvent*):

```
argMainLoop( NULL, keyEvent, mainLoop );
```

Además, debido a la utilización de Allegro, debe invocarse a *END_OF_MAIN()* tras la finalización de la función *mainLoop*:

```
int main(int argc, char** args) {
 ...
}
END_OF_MAIN();
```

La funcion generate_audio, esta función utiliza funciones de Allegro para generar un sonido a partir de un valor recibido como parámetro.

```
static int generate_audio(unsigned char tono) {
```

Inicializamos algunas variables:

```
unsigned char incremento = tono;
unsigned char valor=0x00;
```

A continuación tenemos que inicializar Allegro, un temporizador y el dispositivo de sonido:

```
allegro_init();
install_timer();
install_sound(DIGI_AUTODETECT, MIDI_NONE, NULL);
```

Mediante la función *play_audio_stream* de Allegro creamos un nuevo flujo (stream) de sonido y comenzamos a reproducirlo. Los parámetros que se le pasan a esta función son: tamaño de buffer (número de muestras), número de bits (8 o 16), stereo (TRUE o FALSE), frecuencia de muestreo, volumen (de 0 a 255) y pan (de 0 a 255).

```
stream = play_audio_stream(11025, 8, FALSE, 22050, 255, 128);
```

Una vez tenemos un stream debemos rellenarlo con los valores que queremos reproducir, para ello hay que utilizar la función *get_audio_stream* que devuelve *NULL* si el buffer aun está siendo reproducido, o de lo contrario un puntero a la posición de los datos a reproducir que debemos rellenar .

```
p=(unsigned char*)get_audio_stream_buffer(stream);
```

Si *get_audio_stream* no devuelve *NULL*, utilizamos un bucle para rellenar el valor de cada muestra del buffer (en total 11025), siendo la variable *offset* la posición de la muestra actual. Como *p* es un puntero, podemos acceder a la muestra de la posición *offset* como si fuera un array, mediante *p[offset]*. Los datos a rellenar empiezan en 0 para la primera muestra del buffer y van incrementando en cada iteración a partir de la variable *incremento* que se corresponde con la

variable *tono* recibida, de este modo cuanto mayor sea el valor del tono que se reciba, más agudo será el sonido generado.

```
valor=0x00;
 if (p) {
        for(offset=0;offset<11025;offset++){
            p[offset] = valor;
            valor+=incremento;
        }
 }
```

Una vez se ha rellenado el buffer, debe utilizarse la función *free_audio_stream_buffer* para que el flujo pueda continuar reproduciéndose.

```
free_audio_stream_buffer(stream);
```

Por último almacenamos el tiempo en una variable, lo cual nos sirve para poder controlar desde otra función, que el sonido deje de reproducirse al pasar determinado tiempo. Además tenemos que volver a poner a 0 el *offset*, ya que esta variable está declarada fuera de esta función y si no se pone a 0, conservaría su valor en la siguiente ocasión en que se llame a *generate_audio*.

```
timer=time(NULL);
offset=0;
```

La función guardarEnFichero, es semejante a la de **Configuration**, pero permite guardar información sobre varios patrones en el fichero conf2.con siguiendo la estructura propia de este fichero, para ello se abre el fichro en modo escritura y se utiliza un bucle para escribir los datos contenidos en los arrays.

```
static int guardarEnFichero(void) {
 FILE *fich;
 int i;

 fich=fopen("conf2.con","w");

 if(fich != NULL) {
  fprintf(fich,"%d\n",umbral);
  for(i=0;i<numHSV;i++) fprintf(fich,"#\n%d %d\n%d %d\n%d %d\n",minH[i],maxH[i],minS[i],maxS[i],
minB[i],maxB[i]);
  fprintf(fich,"-");
  fclose(fich);
  return 1;
 }
 return 0;

}
```

La función guardarEnFichero, también es semejante a la de **Configuration**, pero permitiendo que se cargue información sobre varios patrones. Esto se realiza abriendo el fichero en modo lectura y mediante un bucle que va leyendo el fichero y guardando los datos en las variables y arrays correspondientes hasta que se encuentre el carácter de final de fichero.

```
static int cargarDeFichero(void) {
 FILE *fich;
 char c;
 int i=0;
```

```
 fich=fopen("conf2.con","r");

 if(fich != NULL) {
          fscanf(fich,"%d",&umbral);
          fscanf(fich," %c",&c);

          if(c=='#')  //si el separador es correcto y no es fin de fichero inicia bucle
          for(i=0;i<4;i++){

           fscanf(fich,"%d",&minH[i]);
           fscanf(fich,"%d",&maxH[i]);
           fscanf(fich,"%d",&minS[i]);
           fscanf(fich,"%d",&maxS[i]);
           fscanf(fich,"%d",&minB[i]);
           fscanf(fich,"%d",&maxB[i]);
           fscanf(fich," %c",&c);
           if(c!='#') break; //si es el fin del fichero, o el separador no es correcto,
             //finaliza el bucle

          }
  numHSV=i+1;
  fclose(fich);
  return 1;
 }

 return 0;

}
```

La función mainLoop, es la más importante del programa, ya que en ella se detectan los diferentes colores y las colisiones entre ellos, y se realizan las acciones correspondientes. Se comentan los cambios respecto de la función mainLoop de configuration:

Tenemos una variable punto que almacena el punto con que tiene lugar la colisión, y una variable colisión que vale 1 o 0 dependiendo de si ha habido o no una colisión entre el patrón puntero y un patrón objetivo.

```
int          colision=0;
int          punto[2];
```

La función de labeling utilizada es *arLabelingHSBMultiple*, que recibe los mismos parámetros que *arLabelingHSB* a excepción de que los máximos y los mínimos de cada componente son arrays, y se envía un nuevo parámetro *numHSV* que contiene el número total de patrones que deben buscarse en la imagen, es decir, el número de posiciones ocupadas en los arrays.

```
limage = arLabelingHSBMultiple( dataPtr, thresh,
            &label_num, &area, &pos, &clip, &label_ref, 1, minH, maxH, minS,
                  maxS, minB, maxB,numHSV, invert);
```

La comprobación de colisiones tiene lugar en el interior de una serie de bucles. El primero de ellos recorre todos los patrones:

```
for( i = 0; i < marker_num2; i++ ) {
```

En su interior, otro bucle se encarga de dibujar el contorno de cada patrón detectado, igual que

hacíamos en **Configuration**.

```
for(j=0; j<marker2[i].coord_num-1; j++) {
   argLineSeg( marker2[i].x_coord[j] ,  marker2[i].y_coord[j],  marker2[i].x_coord[j+1],
      marker2[i].y_coord[j+1], 0, 0);
 }
```

A continuación, comprobamos de que color es el primer punto del patrón actual, así sabemos si es el patrón puntero o si es alguno de los objetivos. Para ello primero hacemos que una variable *pnt* apunte al primer punto del patrón (teniendo en cuenta que la coordenada *x* del punto se encuentra en *marker2[i].x_coord[0]* y la coordenada *y* en *marker2[i].y_coord[0]*) y luego utilizamos la función *RGBtoHSB*.

```
pnt = (dataPtr+(arImXsize*VALORES_POR_PIXEL*(marker2[i].y_coord[0]-1))+
(marker2[i].x_coord[0]*VALORES_POR_PIXEL));
RGBtoHSV(R_COMP,G_COMP,B_COMP, &H, &S, &V);
```

Ahora que tenemos sus valores en HSV, comprobamos si se encuentran dentro de los rangos del patrón puntero, es decir de los rangos que vienen definidos por las posiciones 0 de los arrays.

```
if((H<maxH[0] && H>minH[0]) && (S<maxS[0] && S>minS[0])
   && (V<maxB[0] && V>minB[0])) {
```

Si esta condición se cumple habremos encontrado el patrón puntero en la posición *i* del array de patrones, de lo contrario seguimos buscando el patrón puntero en la siguiente posición del array. Si hemos encontrado el patrón puntero, debemos comprobar para cada uno de los restantes patrones, si el patrón actual (puntero) colisiona con alguno de ellos (objetivo). Para ello, recorremos mediante otro bucle todos los patrones y comprobamos que no sean del color del puntero ya que no nos interesa la colisión del puntero consigo mismo (en realidad debería bastar con que nos "saltemos" al patrón de la posición *i* puesto que sabemos que ese es el puntero, pero para estar seguros comprobaremos su color ya que en principio nada impide que hayan varios patrones puntero cada uno en una posición distinta).

```
for( k = 0; k < marker_num2; k++ ) {
  pnt =(dataPtr+(arImXsize*VALORES_POR_PIXEL*(marker2[k].y_coord[0]-1))+
(marker2[k].x_coord[0]*VALORES_POR_PIXEL));
  RGBtoHSV(R_COMP,G_COMP,B_COMP, &H, &S, &V);
  //si no es del color del puntero (no es el puntero)
 if(!((H<maxH[0] && H>minH[0]) && (S<maxS[0] && S>minS[0])
         && (V<maxB[0] && V>minB[0]))){
   colision=0;
```

Si no son del color del puntero, utilizamos dos nuevos bucles que recorren cada punto del contorno del patrón puntero (patrón *i*) y del patrón actual (patrón *k*), para comprobar si alguno de los puntos del puntero está suficientemente cerca del otro patrón como para considerarse colisión.En caso de haber colisión, se activa la variable *colision*, y se guarda el punto de colisión de *k,* la coordenada *x* en *punto[0]* y la coordenada *y* en *punto[1]*

```
 for(l=0; l<marker2[k].coord_num-1; l++) { //recorremos los puntos del patrón k
   for(m=0;m<marker2[i].coord_num-1;m++) {//recorremos los del patrón puntero
      if(abs((int)(marker2[i].x_coord[m]-marker2[k].x_coord[l]))<=10
         && abs((int)(marker2[i].y_coord[m]-marker2[k].y_coord[l]))<=10){
      colision=1;//hay colisión
      punto[0]=marker2[k].x_coord[l];
      punto[1]=marker2[k].y_coord[l];
```

```
                    break;
                }//end_if
            }//end_for
        }//end_for
```

En el siguiente paso comprobamos el valor de la variable *colision* para determinar si ha habido o no colisión. Si no la ha habido no hacemos nada, pero en caso de que se haya producido una colisión, primero comprobamos el color del punto de colisión que hemos guardado en *punto,* así sabemos de que patrón se trata y podremos decidir la operación a realizar.

```
    if(colision) {
      pnt = (dataPtr+(arImXsize*VALORES_POR_PIXEL*(punto[1]-1))
            +(punto[0]*VALORES_POR_PIXEL));
     RGBtoHSV(R_COMP,G_COMP,B_COMP, &H, &S, &V);
```

Comprobamos que no esté ya sonando un sonido antes de comenzar a reproducir otro, ya que de lo contrario podrían lanzarse demasiados sonidos identicos a la vez (hay que tener en cuenta que la colisión puede durar varias iteraciones y que en cada una de éstas se estaría volviendo a llamar a la función de reproducir sonido repetidamente si no se comprueba primero que no hay un sonido activo). Si no hay ningún sonido reproduciéndose, entonces comprobamos en cual de todos los rangos encaja el punto de colisión y así sabemos a que patrón le corresponde la colisión. Una vez sabemos con que patrón se produjo la colisión, llamamos a la función *generate_audio* pasándole un valor de tono diferente según el patrón, y activamos la variable *sonando* que indica que hay un sonido activo.

```
            if(!sonando) {
              if(numHSV>1 && (H<maxH[1] && H>minH[1]) && (S<maxS[1]
               && S>minS[1]) && (V<maxB[1] && V>minB[1])) {
                    generate_audio(0x03);sonando=1;
            }
              else if(numHSV>2 && (H<maxH[2] && H>minH[2])
              && (S<maxS[2] && S>minS[2]) &&(V<maxB[2] && V>minB[2]){
                    generate_audio(0x04);sonando=1;
            }
                else if(numHSV>3 && (H<maxH[3] && H>minH[3])
                &&(S<maxS[3] && S>minS[3])&&(V<maxB[3] && V>minB[3]) {
                    generate_audio(0x05);sonando=1;}
                      else {generate_audio(0x06);sonando=1;}
              }//end_if
            }//end_if
           }//end_if
        }//end_for
      }//end_if
    }//end_for
```

Comprobando primero el valor de la variable *numHSV* se evita acceder a direcciones de memoria que no han sido inicializadas, ya que si solo tenemos tres patrones objetivo (*numHSV = 3*), no tiene sentido comprobar si la colisión se ha producido con el cuarto patrón objetivo.

Con ésto han terminado todos los bucles. Ahora solo nos queda comprobar si hay un sonido activo y si éste ya ha durado uno o más segundos, en cuyo caso lo detenemos y desactivamos la variable *sonando* para que puedan reproducirse más sonidos con las siguientes colisiones. Además dejamos el *timer* a 0 y cerramos la ventana de Allegro.

```
    if(sonando && time(NULL)-timer>=1)  {
```

```
    stop_audio_stream(stream);
    timer=0;
    sonando=0;
    allegro_exit();
}
```

5.4 Redefiniendo el click del ratón

En este ejemplo se muestra como puede simularse un *click* con el patrón. Consiste en "hacer desaparecer" el patrón durante un tiempo determinado, es decir hay que taparlo y destaparlo rápidamente. En el ejemplo, si se hace *click* mientras el patrón se encuentra sobre la pelota, comienza a arrastrarse la pelota con el movimiento del patrón, y se suelta al hacer *click* de nuevo.

La mayor parte de las funciones de conservan del programa **Configuration**, la diferencia es que en la función mainLoop se han añadido operaciones para detectar el *click* y arrastrar una pelotita. Además se ha añadido una función para dibujar una pelotita en 2D.

La función dibujaPelota, se encarga de dibujar la pelotita en cada frame en la posición (*cx,cy*) que le corresponda. Se ha dibujado en 2D utilizando openGL, mediante una aproximación a un círculo realizada mediante un polígono de 100 lados, donde *r* es el radio. Las variables *cx* y *cy* indican el centro del círculo que va cambiando conforme la pelotita es arrastrada por la pantalla.

```
static void dibujaPelota() {
   int i;
   /* inicializa los valores de la vista */
   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
   gluOrtho2D(0.0, 1.0, 0.0, 1.0);
   glColor3f(0, 0, 1.0);
   glBegin(GL_POLYGON);
   for (i = 0; i < 100 + 1; i++) { // +1 para cerrar
     glVertex2f( cx + r * cos(2.0 * 3.14159 * i / 100),
                 cy + r * sin(2.0 * 3.14159 * i / 100) );
    }
   glEnd();
   /* Vacia el buffer de dibujo */
   glFlush ();
}
```

La función mainLoop, después de la invocación a *arLabelingHSB* y a *arDetectMarker3*, ya tenemos los patrones detectados en la imagen. Por simplificar, se ha supuesto que el usuario se encargará de cuidar que el color del patrón no se repita en la escena.

Debemos ahora detectar si se produce algún *click*, lo cual quiere decir que el patrón tiene que desaparecer y volver a aparecer en un cierto tiempo. Es decir para que se detecte un *click*, tienen que sucederse la siguiente secuencia ordenada de pasos:

1. Se detecta el patrón en la imagen y el timer se pone a 0
2. De repente ya no se encuentra el patrón en la imagen, el timer guarda el tiempo actual
3. Antes de que pase el tiempo límite vuelve a detectarse y el timer se pone a 0

Veamos el algoritmo utilizado:

No se detecta el patrón en la imagen ya que *marker_num2* = 0 (situación 2), el *timer* vale 0 por tanto se ha detectado un patrón en la iteración anterior y por eso se ha puesto a 0 (situación 1). Se han cumplido los pasos 1 y 2, ahota hay que contar el tiempo que pasa desde este momento (acaba de desaparecer el patrón) hasta que vuelva a aparecer, para ello guardamos el tiempo actual en la variable *timer*.

```
if(timer==0 && marker_num2==0) {
 click=0;
 timer = (long) time(NULL);
}
```

El tiempo es mayor que 0 (por tanto hemos pasado por la situación 1 y 2), y además se ha encontrado el patrón en esta iteración, por tanto solo nos falta comprobar si se ha superado el tiempo límite. Restamos el valor del *timer* al tiempo actual, si todavía no ha pasado 1 segundo (tiempo límite) se ha producido un *click* (situación 3). De lo contrario se ha detectado el objeto y el timer se pone a 0 (situación 1)

```
else if (timer>0 && marker_num2>0){
  if((time(NULL)-timer)<1){click=1;}
    timer = 0;
}
```

En cualquier otro caso, no ha habido *click*.

```
else {
 click=0;
}
```

Una vez que hemos comprobado si ha habido *click*, dibujamos el contorno del patrón y comprobamos si ha habido colisión. Para ello iniciamos un bucle en el que recorremos todos los patrones detectados (aunque en principio solo hay uno).

```
for( i = 0; i < marker_num2; i++ ) {
```

Para cada uno de ellos, recorremos todos los puntos del patrón y dibujamos su contorno.

```
for(j=0; j<marker2[i].coord_num-1; j++) {
   argLineSeg( marker2[i].x_coord[j] ,  marker2[i].y_coord[j],  marker2[i].x_coord[j+1],
     marker2[i].y_coord[j+1], 0, 0);
```

Aprovechando el mismo bucle, y en caso de que se haya realizado un *click* en esta iteración, comprobamos si alguno de los puntos del contorno está dentro del círculo que representa la pelotita. Hay que tener en cuenta que las coordenadas *cx* y *cy,* y el radio *r* del círculo, tienen valores normalizados, es decir, valores del 0 al 1, para convertirlos a valores de la pantalla tenemos que multiplicarlos por *arImXSize* o *arImYSize* según corresponda. Además, puesto que en coordenadas normalizadas los valores de *y* están invertidos respecto de las coordenadas de pantalla de ARToolKit, hay que restarle a *arImYsize* el valor obtenido para obtener el auténtico valor de la coordenada *y* en coordenadas de pantalla .Si el valor absoluto de la diferencia entre la coordenada *x* del punto actual del patrón (punto *j*) y la posición x del centro de la circunferencia (*cx*), es menor que el radio, entonces el punto cumple la condición de interioridad para la coordenada *x*. Si además esto se cumple con la coordenada *y*, podemos asegurar que hay algún punto del puntero

que está dentro del círculo y que, por tanto, colisiona con él. Si esta colisión se produce, almacenamos el punto del patrón puntero con el que se ha producido la colisión, cambiamos el valor de *arrastrar* a 1 si valia 0, o a 0 si vaía 1 (esta variable indicar si se debe arrastrar la pelotita o no, por lo tanto debe activarse cuando se realice un *click* dentro de la pelotita y desactivarse cuando vuelva a realizarse un click dentro de la pelota) y finalizamos el bucle, ya que nos basta con que uno solo de los puntos colisione para saber que ha habido colisión, no es necesario comprobar todos los demás.

```
if(click) {
  if(abs((int)(marker2[i].x_coord[j]-(cx*arImXsize)))<(r*arImXsize)
  && abs((int)(marker2[i].y_coord[j]-(arImYsize-(cy*arImYsize))))<(r*arImYsize)){
          arrastrar=!arrastrar;
          punto[0]=marker2[i].x_coord[j];
          punto[1]=marker2[i].y_coord[j];
          break;
   }//end_if
```

Podemos añadir una sentencia *else*que permite que deje de arrastrarse también si se realiza *click* en cualquier parte (no necesariamente dentro de la pelotita), pero como la pelotita estará siendo arrastrada, siempre estará colisionando con el puntero a menos que el puntero se salga de la pantalla.

```
    else {
     arrastrar=0;
    }
   }//end_if
  }//end_for
}//end_for
```

Si la variable *arrastrar* está activada, la pelotita debe seguir al patrón. Para ello lo que se hace es asignar al centro de la pelotita el valor del centro del patrón (*marker2[0].pos[0] y marker2[0].pos [1]*), y transformarlo de nuevo a coordenadas normalizadas dividiendo entre *arImXsize* o *arImYsize*. Además para el caso de la coordenada *y*, hay que tener en cuenta que en coordenadas normalizadas sus valores se encuentran invertidos respecto de las coordenadas de la pantalla de ARToolKit, por tanto hay que restarle a 1.0 el resultado obtenido para obtener *cy*.

```
    if(arrastrar) {
     cx=(double)marker2[0].pos[0]/(double)arImXsize;
     cy=1.0-((double)marker2[0].pos[1]/(double)arImYsize);
    }
```

Por último se dibuja la pelota en las nuevas coordenadas

```
    dibujaPelota();
    argSwapBuffers();
}
```

5.5 Interacción con objetos virtuales: El juego de la pelotita

Es un ejemplo algo más avanzado que los anteriores, que muestra como se puede implementar un juego simple de Realidad Aumentada, en el que un patrón de color simula un joystick sencillo. Este programa ha ido evolucionando desde una primera versión hasta una cuarta versión, por tanto explicaremos la primera versión y luego comentaremos los cambios realizados y características

añadidas a las sucesivas versiones, además algunas partes del programa son iguales a las del programa **Configuration** y no necesitan volver a explicarse. La primera y segunda versión se corresponden al programa **Sample**, mientras que las otras dos se corresponden con el programa **Pelotita**.

Primera versión: la pelotita rebota contra el patrón elegido como si éste fuera una especie de raqueta. Para ello se comprueba que la pelota colisione con algún punto del patrón.

Tenemos definidas las siguientes variables que determinan la posición, radio, velocidad y dirección de la pelotita. A parte tenemos dos variables más, *dentro* que se activa cuando algún punto del patrón colisiona con la pelota, y *punto* que almacena las coordenadas del punto de colisión.

```
double cx=0.5;
double cy=0.5;
double velx;
double vely;
int dirx;
int diry;
double r=0.02;
int dentro = 0;
int punto[2];
```

La declaración de cabeceras de funciones es la que sigue:

```
static void   init(void);
static void   cleanup(void);
static void   keyEvent( unsigned char key, int x, int y);
static void   mouseEvent(int button, int state, int x, int y);
static void   mainLoop(void);
static int    guardarEnFichero(void);
static int    cargarDeFichero(void);
static void   dibujaPelota(void);
```

La función cleanup, no presenta variaciones respecto de la del programa **Configuration**.

La función init, es parecida a la del programa configuration, pero presenta la diferencia de que al inicio de esta función, se dan valores iniciales a las variables relacionadas con la velocidad (en coordenadas normalizadas de 0 a 1) y dirección de la pelota., teniendo en cuenta que un valor de 1 en la dirección de una coordenada, significa que la pelota se desplaza en el sentido en que esa coordenada aumenta, y un valor de -1 significa que la pelota se desplaza en sentido inverso en esa coordenada. Además la dirección inicial en ambas coordenadas es aleatoria.

```
srand( (unsigned int) time( NULL ) );
velx=0.5;
vely=1;
dirx=1;diry=1;
if(rand()%2<1) dirx=-1;
if(rand()%2<1) diry=-1;
```

Las funciones cargarDeFichero y guardarEnFichero se conservan del programa **Configuration**, sin ninguna variación.

La función dibujaPelota, sirve para dibujar la pelota en cada iteración en su nueva posición (*cx* y *xy*) y es exactamente la misma que la explicada anteriormente en el programa **Clicksample**.

La función mouseEvent, se encarga de manejar los eventos de ratón, conservándose del programa **Configuration**.

La función keyEvent, se encarga de manejar los eventos de teclado. Es la misma que la del programa **Configuration**, pero se le ha añadido un nuevo evento sobre la tecla 'r' que permite al usuario reiniciar el juego (la pelota vuelve a la posición central de la pantalla con la velocidad inicial y con direccion aleatoria).

```
if( key == 'r' ) {
        cx=0.5;
        cy=0.5;
        srand( (unsigned int) time( NULL ) );
       velx=0.5;
        vely=1;
        accx=0;
        accy=0;
        dirx=1;diry=1;
        if(rand()%2<1) dirx=-1;
        if(rand()%2<1) diry=-1;
}
```

La función main, tampoco presenta cambios respecto a la de **Configuration**.

La función mainLoop, implementa el bucle principal del programa. Además de detectar los patrones de la imagen, se encarga de comprobar si existe alguna colisión entre el patrón con la pelota, y en su caso actualizar los valores de la pelota para que "rebote".

Para conseguir esto implementamos un bucle que recorre el array de patrones detectados (aunque en principio solo hay uno, nada impide que puedan haber varias "raquetas" en pantalla).

```
for( i = 0; i < marker_num2; i++ ) {
```

En su interior, un nuevo bucle recorre cada uno de los puntos del contorno del patrón para dibujarlo.

```
for(j=0; j<marker2[i].coord_num-1; j++) {
      argLineSeg( marker2[i].x_coord[j] ,  marker2[i].y_coord[j],
                  marker2[i].x_coord[j+1],  marker2[i].y_coord[j+1], 0, 0);
```

Aprovechando este bucle, se comprueba la distancia entre el punto del patrón y el centro de la pelota, si esta distancia es menor que el radio, el punto está dentro de la pelota y por tanto hay colisión. En tal caso, activamos la variable *dentro* y guardamos el punto de colisión.

```
 if(abs((int)(marker2[i].x_coord[j]-(cx*arImXsize)))<(r*arImXsize)
   && abs((int)(marker2[i].y_coord[j]-(arImYsize-(cy*arImYsize))))<(r*arImYsize)){
       dentro=1;
       punto[0]=marker2[i].x_coord[j];
       punto[1]=marker2[i].y_coord[j];
  }//end_if
}//end_for
```

Fuera de este bucle, se comprueba si la variable *dentro* está activa (al menos uno de los puntos está dentro, por tanto ha habido colisión) y si se da el caso, se actualizan las variables de la pelotita

mediante un cambio de dirección (rebote). Para calcular el cambio de dirección, se utiliza la posición del punto de colisión almacenado en *punto* de modo que si está a la derecha del centro del radio, la pelotita se moverá hacia la izquierda, de lo contrario se moverá hacia la derecha. Si el punto de colisión está por debajo del centro de la pelota, la pelota se moverá hacia arriba, en caso contrario se moverá hacia abajo.

```
 if(dentro) {
      if(punto[0]>cx*arImXsize) {dirx=-1;}
        else {dirx=1;}
      if(punto[1]>(arImYsize-cy*arImYsize)) {dirx=-1;}
        else {diry=1;}
 }//end_if
}//end_for
```

A continuación, ya fuera de todos los bucles, hay que actualizar la posición de la pelotita (independientemente de que haya o no haya colisión). Su nueva posición se consigue sumándole o restándole (dependiendo de la dirección) la velocidad respecto de su posición actual.

```
 cx+=velx*dirx;
 cy+=vely*diry;
```

Hay que comprobar también que la pelota solo pueda salir de la pantalla por la parte inferior. Para ello se comprueba que la posición de la pelota no supere los límites derecho, izquierdo y superior de la pantalla, y en caso de producirse se produce un cambio de dirección para que la pelota rebote en las esquinas.

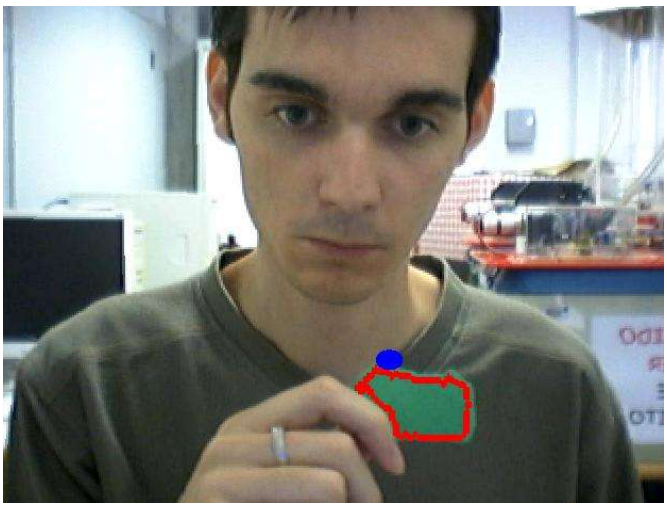
```
 if((cx*arImXsize+(arImXsize*r))>=arImXsize && dirx==1) {dirx=-1;}
 if((cx*arImXsize-(arImXsize*r))<=0 && dirx==-1) {dirx=1;}
 if((cy*arImYsize+(arImYsize*r))>=arImYsize && diry==1) diry=-1;
```

Si la pelota cae por el limite inferior, el jugador ha perdido y el juego se reinicia volviendo la pelotita a su posición inicial.

```
 else if((cy*arImYsize-(arImYsize*r))<=0) {cx=0.5;cy=0.5;}
```

Por último se dibuja la pelota en su nueva posición mediante la función *dibujaPelota*.

```
 dibujaPelota();
 argSwapBuffers();
}
```



Segunda versión: Es el mismo programa pero añadiendo una serie de modificaciones que permiten simular un efecto de aceleración con cada rebote, es decir, cada vez que rebota la pelota se mueve más rápido, aumentando asi la dificultad del juego. Tenemos declaradas, además de las variables de velocidad, dos variables que controlan la aceleración de la pelota.

```
double accx, accy;
```

Se han definido los valores máximos de aceleración y velocidad, con el fin de evitar que la pelota se mueva tan rápido que "atraviese" al patrón raqueta (puesto que en cada iteración la posición se desplaza según la velocidad, si la velocidad es demasiado grande acabará saltándose al patrón sin tocarlo).

```
#define MAX_VEL 3*r
#define MAX_ACC 0.007
```

La función init,cambia ligeramente, mediante una reducción de la velocidad (valores muy grandes de velocidad hacen el juego injugable, más aun cuando ésta aumenta continuamente debido a al aceleración) y la inicialización de la aceleración a 0 (no se acelera hasta que rebota por primera vez).

```
srand( (unsigned int) time( NULL ) );
velx=0.01;
vely=0.02;
accx=0;
accy=0;
dirx=1;diry=1;
if(rand()%2<1) dirx=-1;
if(rand()%2<1) diry=-1;
```

La función keyevent, también cambia ligeramente. El evento para la tecla 'r' incluye ahora una reinicialización de los valores de aceleraciíon, además de la reinicialización a una velocidad más baja.

```
if( key == 'r' ) {
        cx=0.5;
        cy=0.5;
        srand( (unsigned int) time( NULL ) );
        velx=0.01;
        vely=0.02;
        accx=0;
```

```
        accy=0;
        dirx=1;diry=1;
        if(rand()%2<1) dirx=-1;
        if(rand()%2<1) diry=-1;
}
```

En la función mainLoop, se han realizado modificaciones que permiten que la aceleración aumente cada vez que se produce un rebote contra el patrón a razón de 1.01 (un valor muy pequeño para evitar que se alcance enseguida el valor máximo). Si es el primer rebote (todavía no existía aceleración) se inicializa la acceleración a un valor de 0.001.

```
if(dentro) {
  if(punto[0]>cx*arImXsize) {dirx=-1;}
   else {dirx=1;}
  if(punto[1]>(arImYsize-cy*arImYsize)) {dirx=-1;}
    else {diry=1;}

  if(accx>0 && accy>0) {accx=accx*1.01; accy=accy*1.01;}
      else if(accx<=MAX_ACC && accy<=MAX_ACC) {
          accy=0.001; accy=0.001;
      }
   }//end_if
```

Posteriormente, y antes de actualizar la posición de la pelotita, se procede a actualizar la velocidad de la pelota sumándole la aceleración en cada iteración, siempre y cuando el valor de velocidad no exceda el máximo.

```
if(velx<=MAX_VEL){
 velx+=accx;
}
if(vely<=MAX_VEL){
 vely+=accy;
}
```

Después de esto ya se actualiza la posición del mismo modo que se hacía antes, sumando o restando (según la dirección) la velocidad respecto de la posición actual de la pelota.

```
cx+=velx*dirx;
cy+=vely*diry;
```

Un último cambio se produce cuando la pelota atraviesa el límite inferior de la pantalla, en tal caso, hay que ponr la aceleración a 0 y la velocidad a su valor inicial.

```
else if((cy*arImYsize-(arImYsize*r))<=0) {
      cx=0.5;cy=0.5; accx=0; accy=0; velx=0.01, vely=0.02;
}
```

Tercera versión: Esta versión presenta bastantes cambios respecto de las anteriores, hasta el punto que es considerada como un programa distinto, el programa **Pelotita**. Básicamente las modificaciones sobre versiones anteriores consiste en:

- Quitar la imagen de vídeo de la cámara.
- El jugador ahora viene representado en pantalla mediante un rectángulo que se mueve utilizando el patrón como si fuera un joystick.
- Las colisiones ahora se realizan respecto del rectángulo, que tiene una forma y tamaño

constantes, lo cual es mucho más eficiente que comprobar cada punto del contorno del patrón.

Quitar la imagen de vídeo:

Para quitar la imagen de vídeo, debemos modificar la función mainLoop, cambiando las siguientes líneas:

```
argDrawMode2D();
argDispImage( dataPtr, 0,0 );
```

Por ésta:

```
glClear (GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
```

De éste modo evitamos que se muestre la imagen, y con la línea que hemos añadido, conseguimos que se borre la pantalla en cada iteración para actualizar la posición de los objetos.

Dibujar un rectángulo que representa al jugador:

Se han definido los valores de la altura y anchura del rectángulo utilizado.

```
#define PLAYER_HEIGHT  0.04
#define PLAYER_WIDTH  0.12
```

Se ha creado una nueva función, *dibujaJugador*, que se encarga de dibujar al jugador en su nueva posición (que recibe como parámetro) en cada iteración, siguiendo el movimiento del patrón. El jugador viene representado como un cuadrado (poligono de 4 lados) que se dibuja, utilizando coordenadas normalizadas entre 0 y 1, a partir de su centro (*x*, *y*) utilizando restando y sumando al centro los valores de la mitad de anchura y altura para obtener sus cuatro esquinas.

```
static void dibujaJugador(double x, double y) {
/* inicializa los valores de la vista */
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluOrtho2D(0.0, 1.0, 0.0, 1.0);
glColor4f(0,0,0,1.0f);
glBegin(GL_POLYGON);
  glVertex2f(x-PLAYER_WIDTH/2,y-PLAYER_HEIGHT/2);
  glVertex2f(x-PLAYER_WIDTH/2,y+PLAYER_HEIGHT/2);
  glVertex2f(x+PLAYER_WIDTH/2,y+PLAYER_HEIGHT/2);
  glVertex2f(x+PLAYER_WIDTH/2,y-PLAYER_HEIGHT/2);
glEnd();

/* Vacia el buffer de dibujo */
glFlush ();
}
```

En la función *mainLoop* ya no se dibuja el contorno de los patrones detectados, en lugar de ello, se invoca a la función *dibujaJugador* para que el jugador se dibuje siempre en su nueva posición. Para ello se recorre el array de patrones y se dibuja un jugador por cada patrón encontrado utilizando como centro del rectángulo, el centro del patrón (*marker2[i].pos[0]* y *marker2[i].pos [1]*). De este modo el jugador siempre se movera siguiendo al patrón.

```
for( i = 0; i < marker_num2; i++ ) {
```

```
dibujaJugador(marker2[i].pos[0]/arImXsize,1-marker2[i].pos[1]/arImYsize);
```

Comprobar colisiones de la pelota con el rectángulo:

Se ha creado una función, *colisionJugador*, que permite comprobar si ha tenido lugar una colisión, y en tal caso, si dicha colisión se ha producido por arriba, por abajo, por la izquierda o por la derecha, con la intención de poder utilizar esta información para cambiar la dirección de la pelota.

La función recibe como parámetros las coordenadas del centro del jugador, las del centro de la pelota y el radio de ésta.

```
static int colisionJugador(double jx, double jy, double px, double py, double prad) {
```

Estas coordenadas se multiplican por 100 debido a que la función de valor absoluto (*abs*) solo funciona con números enteros, así que para poder utilizarla con números reales nos vemos obligados a realizar los cálculos con valores del 0 al 100 en lugar de valores del 0 al 1. La comparación entre la distancia del centro de la pelota y el centro del rectángulo se realiza por separado para las coordenadas *x* e *y*.

```
 double compx;
 double compy;
jx*=100;
 jy*=100;
 px*=100;
 py*=100;
 prad*=100;
 compx = abs((int)px-(int)jx);
 compy = abs((int)py-(int)jy);
```

Las variables *compx* y *compy,* contienen la distancia entre los centros de ambas figuras en *x* y en *y* respectivamente. Ahora lo que debemos hacer, es comprobar a partir de estas distancias, si las dos figuras están en contacto. Para que haya colisión en un coordenada, se debe cumplir que la distancia calculada en esa coordenada sea menor que la suma entre la mitad del lado del rectángulo en la coordenada elegida y el doble del radio.

```
 if(compx <= 2*prad+PLAYER_WIDTH*100/2
    && compy <= 2*prad+PLAYER_HEIGHT*100/2){
```

Si existe colisión se devuelve un número del 1 al 4 que indica en que dirección se produjo la colisión. Esta dirección se calcula comparando la posición del centro de la pelota respecto del centro del rectángulo

```
//1 colision por la derecha-arriba
 //2 colision derecha-abajo
 //3 colision izquierda-arriba
 //4 colision izquierda-abajo

        if(px>=jx && py>=jy) return 1;
        else if(px>=jx && py<jy) return 2;
        else if(px<jx && py>=jy) return 3;
        else  return 4;
}
```

Si no existe colisión el valor devuelto es 0.

```
        //0 no colision
        return 0;
    }
```

Esta función, se utiliza en *mainLoop* después de dibujar el jugador, para comprobar en cada iteración si se produce colisión, y en caso de producirse se procede a cambiar la dirección de la pelota para que rebote y a incrementar la aceleración.

```
for( i = 0; i < marker_num2; i++ ) {
   dibujaJugador(marker2[i].pos[0]/arImXsize,1-marker2[i].pos[1]/arImYsize);
   direccion = colisionJugador(marker2[i].pos[0]/arImXsize,1-marker2[i].pos[1]/arImYsize,cx,cy,r);
```

Se comprueba la dirección y se cambian las variables de dirección según corresponda para que la pelota se mueva en dirección contraria a la dirección de colisión.

```
switch(direccion) {
      case 1: dirx=-1; diry=1; break;
      case 2: dirx=-1; diry=-1; break;
      case 3: dirx=1; diry=1; break;
      case 4: dirx=1; diry=-1; break;
      case 0: break;
}
```

Además, si hubo colisión y estamos en Windows, se emite un pitido.

```
                    if(direccion!=0){

                      #ifdef _WIN32
                              Beep(500,50);
                    #endif
```

La aceleración se pone a 0.001 si estaba aun a 0, y incrementa a razón de 1.03 con cada colisión. Se aumentan los puntos y se muestran por consola.

```
    if(accx==0 || accy==0) {accx=0.001; accy=0.001;}
          accx=accx*(double)1.01; accy=accy*(double)1.03;
          puntos++;
          printf("\nPuntos: %d Anterior puntuacion: %d\n",puntos,antPuntos);
    }
}
```

Otras modificaciones:

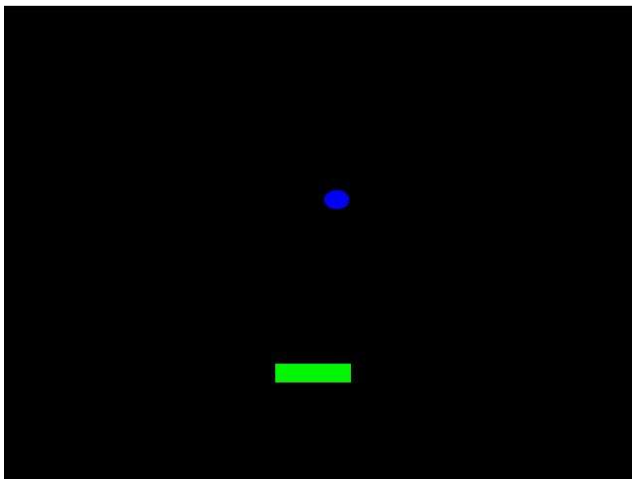
Se han utilizado otros valores de velocidad inicial y aceleración más adecuados. El aumento de velocidad de la función *mainLoop* se produce de forma diferente:

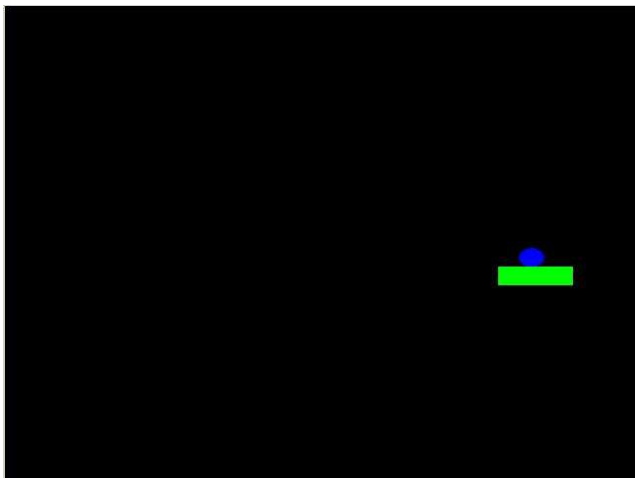
```
if(velx<=MAX_VEL){
 velx+=accx/100.0;
}
if(vely<=MAX_VEL){
 vely+=accy/100.0;
}
```

Se emite un pitido agudo, si el sistema utilizado es Windows, cuando la pelota rebota contra las esquinas, y uno más grave cuando se sale por la parte inferior. Además se lleva la cuenta de los

puntos.

```
cx+=velx*dirx;
cy+=vely*diry;
if((cx*arImXsize+(arImXsize*r))>=arImXsize && dirx==1) {dirx=-1;
   #ifdef _WIN32
    Beep(300,50);
  #endif
}
else if((cx*arImXsize-(arImXsize*r))<=0 && dirx==-1) {dirx=1;
   #ifdef _WIN32
    Beep(300,50);
  #endif
}
else if((cy*arImYsize+(arImYsize*r))>=arImYsize && diry==1) {diry=-1;
 #ifdef _WIN32
    Beep(300,50);
#endif
}
else if((cy*arImYsize-(arImYsize*r))<=0) {
  cx=0.5;cy=0.5; accx=0; accy=0; velx=0.01, vely=0.03;
 antPuntos=puntos; puntos=0;
  #ifdef _WIN32
          Beep(100,700);
  #endif
}
```





Cuarta versión: En esta última versión, se han añadido texturas de imagenes al jugador y la pelota con el fin de hacerlo más atractivo. Se comentan a continuación los cambios necesarios para poner textura a estas dos figuras.

Se han añadido dos funciones nuevas. Una de ellas, *cargarTGA,* nos permite cargar una imagen de tipo TGA y almacenarla en una estructura de tipo *textura*. Otra función, *cambiarTamano*, nos permite cambiar que la ventana del programa pueda cambiar de tamaño sin que esto afecte al correcto funcionamiento del programa. Tanto la estructura *textura*, como las dos funciones nombradas, han sido extraidas del articulo de la siguiente página:

http://articulos.conclase.net/glut/glut004.html

Estructura textura

Permite almacenar los datos de la imagen, bits por pixel, tamaño y un identificador.

```
typedef struct {
   GLubyte  *dibujo;
   GLuint      bpp;
   GLuint largo;
   GLuint ancho;
   GLuint ID;
}textura;
```

Función cargarTGA

La función cargarTGA es demasiado extensa para incluirla aquí, puede verse en la página web anteriormente citada o en el código fuente del fichero incluido en los anexos. Lo que nos interesa saber de esta función, es que a partir de un fichero de imagen (cuyo nombre recibe como parámetro), rellena una estructura de tipo textura (que también recibe como parámetro) con los datos de la imagen.

Función cambiarTamano

Actualiza los parámetros de openGL para que los elementos de la ventana se adapten al nuevo tamaño de ésta. Esta función se invoca siempre que se redimensione la ventana.

```
void cambiarTamano(int largo, int ancho) {
      if(ancho==0) ancho=1;          // Previene que dividamos por 0
      glMatrixMode(GL_PROJECTION);    // Escojemos la matriz de proyeccion
      glLoadIdentity();              // Se resetea la matriz
      glViewport(0,0,largo, ancho);   // Se va a usar toda la ventana para mostrar graficos
      gluPerspective( 45 ,           // Angulo de vision
                  (float)largo/(float)ancho,
                  1.0,             // Cuan cerca se puede ver
                  1000);           // Cuan lejos se puede ver
      glMatrixMode(GL_MODELVIEW);     // Escojemos la matriz de vista
         glLoadIdentity();                    // Se resetea la matriz
      gluLookAt( 0.0, 0.0, 0.0,       // Hacia donde miramos
               0.0, 0.0, -3.0,      // Desde donde miramos
               0.0, 1.0, 0.0);      // Que eje es el que esta hacia arriba
}
```

Para utilizar estos elementos en nuestro programa, hemos creado un array de dos posiciones que va a contener las texturas de los dos objetos.

```
textura  texturas[2];
```

En al función *init* se añaden las siguientes líneas.

```
glEnable(GL_TEXTURE_2D);
      #ifdef _WIN32
        if(!cargarTGA("images\\pelota.tga", &texturas[0])
                        || !cargarTGA("images\\jugador.tga", &texturas[1])) {
      #else
       if(!cargarTGA("images/pelota.tga", &texturas[0])
                       || !cargarTGA("images/jugador.tga", &texturas[1])) {
      #endif
        printf("Error cargando textura\n");
```

```
        exit(1);
    }
```

La primera de ellas sirve para que se activen las texturas. Luego se utiliza la función *cargarTGA* para cargar cada una de las imágenes que necesitamos en su correspondiente posición del array *texturas*. Se ha hecho de forma separada para Windows y para Linux debido a que en linux el carácter separdor es "/" y en windows es "\". Si alguna de las dos texturas no se puede cargar, el programa finaliza.

Una vez que se han cargado las texturas, tenemos que ponérselas a los objetos, para ellos tenemos que modificar las funciones *dibujarJugador* y *dibujarPelota*. Comenzamos con *dibujarJugador*.

Por un lado se ha añadido una llamada a *glBindTexture* que indica a openGL que tiene que utilizar una textura para pintar el objeto, y cual es el identidicador de ésta. Por otro lado, dentro de la clausula *glBegin*, se ha indicado para cada vertice del rectángulo, el punto de la textura que le corresponde mediante la función *glTexCoord2f*.

```
    static void dibujaJugador(double x, double y) {

        /* inicializa los valores de la vista */
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        gluOrtho2D(0.0, 1.0, 0.0, 1.0);

        glBindTexture(GL_TEXTURE_2D,texturas[1].ID);

        glBegin(GL_POLYGON);
          glTexCoord2f(0.0,0.0); glVertex2f(x-PLAYER_WIDTH/2,y-PLAYER_HEIGHT/2);
          glTexCoord2f(0.0,1.0); glVertex2f(x-PLAYER_WIDTH/2,y+PLAYER_HEIGHT/2);
          glTexCoord2f(1.0,1.0); glVertex2f(x+PLAYER_WIDTH/2,y+PLAYER_HEIGHT/2);
          glTexCoord2f(1.0,0.0); glVertex2f(x+PLAYER_WIDTH/2,y-PLAYER_HEIGHT/2);

        glEnd();

        /* Vacia el buffer de dibujo */
        glFlush ();
    }
```

En cuanto a la función *dibujaPelota*, para poder ponerle textura hemos dejado de dibujarla como un circulo y en lugar de eso la hemos dibujado como un rectángulo y le hemos puesto una textura de una pelota sobre fondo negro (el mismo fondo que la pantalla). El cuadrado se ha dibujado de forma parecida al rectángulo pero utilizando el radio como tamaño del lado del cuadrado. Igual que en el caso del jugador, ha habido que llamar a la función *glBindTexture* y asignar la posición de la textura mediante gltexCoord2f.

```
    static void dibujaPelota() {
      /* inicializa los valores de la vista */
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        gluOrtho2D(0.0, 1.0, 0.0, 1.0);

        glBindTexture(GL_TEXTURE_2D,texturas[0].ID);

        //En lugar de dibujar un circulo, dibujamos un cuadrado y le pegamos la textura de la pelota,
        como tiene un canal alpha para transparencias parece un circulo
        glBegin(GL_POLYGON);
```

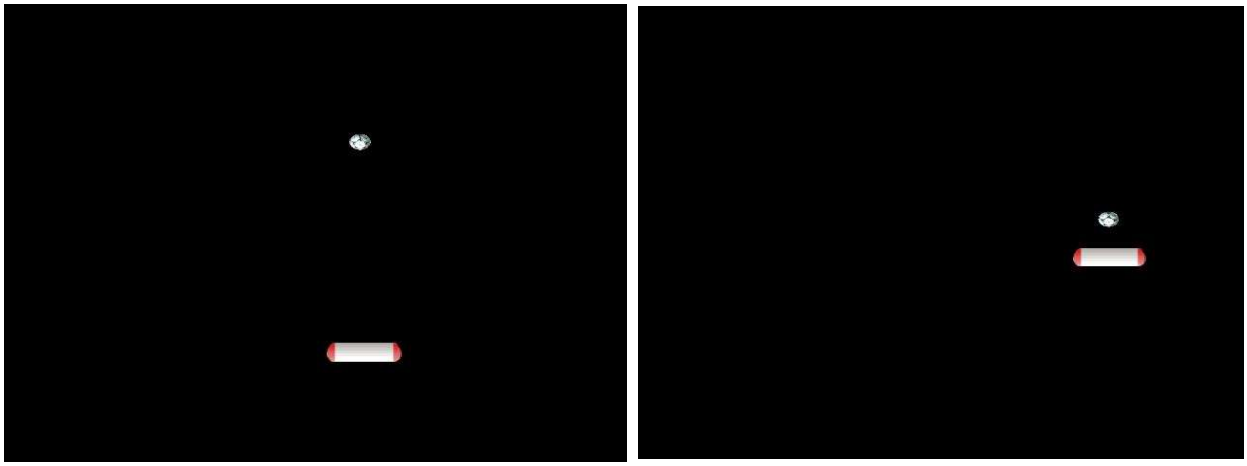
```
    glTexCoord2f(0.0,0.0); glVertex2f(cx-r,cy-r);
    glTexCoord2f(0.0,1.0); glVertex2f(cx-r,cy+r);
    glTexCoord2f(1.0,1.0); glVertex2f(cx+r,cy+r);
    glTexCoord2f(1.0,0.0); glVertex2f(cx+r,cy-r);

  glEnd();
  glFlush();

  /* Vacia el buffer de dibujo */
  glFlush ();

}
```

Hay que tener en cuenta que para que la función cargarTGA funcione correctamente, la imagen debe tener formato TGA sin ningún tipo de compresión, además debe ser cuadrada y su tamaño debe ser potencia de 2, de lo contrario la función no carga la imagen.



5.6 Interacción con objetos virtuales: Un puzzle

Esta aplicación, es un puzzle de cuatro piezas que el usuario debe montar. Las piezas se seleccionan y arrastran utilizando el patrón elegido (representado por un puntero). El usuario no ve la imagen de vídeo de la cámara, solo las piezas y el puntero.

En este ejemplo, de nuevo se utiliza la estructura de tipo textura. Además hemos creado mediante una estructura, un tipo de objeto al que hemos llamado *objetoDibujable*, y a partir de éste hemos creado los tipos *pieza* y *pointer*, a los cuales pertenecen las piezas y el puntero respectivamente.

Todo objeto dibujable tiene una posición (*x* e *y*), un tamaño, una variable que indica si está siendo arrastrada por el puntero (en el caso de las piezas) o si está arrastrando a alguna pieza (en el caso del puntero), una imagen y una máscara (ésta variable solo la usa el puntero).

```
typedef struct {
  double posx;
  double posy;
  double tamx;
  double tamy;
  int arrastrar;
  textura text;
  textura mask;
} objetoDibujable;
```

```
typedef objetoDibujable pieza;
typedef objetoDibujable pointer;
```

Creamos los objetos que vamos a necesitar. Utilizaremos cuatro objetos de tipo pieza, ya que el puzzle se compone de cuatro piezas. Necesitaremos siete texturas, cuatro de ellas para las piezas y tres para el puntero (puede tomar hasta tres formas dependiendo de su estado). Tres máscaras (una para cada imagen del puntero) que servirán para crear efecto de transparencia, como veremos más adelante. Y un objeto de tipo *pointer* que será nuestro puntero.

```
pieza piezas[4];
textura texturas[7];
textura mascaras[3];
pointer puntero;
```

La declaración de cabeceras es la siguiente:

```
static void   init(void);
static void   cleanup(void);
static void   keyEvent( unsigned char key, int x, int y);
static void   mainLoop(void);
static int    guardarEnFichero(void);
static int    cargarDeFichero(void);
static void creaPiezas();
static void dibujaPiezas(int pos);
static void creaPuntero();
static void dibujaPuntero();
static int comprobarColision();
static int comprobarPiezas(int pos);
int cargarTGA( char *nombre, textura *imagen) ;
void cambiarTamano(int largo, int ancho);
```

Las funciones *cleanup*, *keyEvent, guardarEnFichero*, *cargarDeFichero*, *cargarTGA, cambiarTamano y main* ya se han explicado en programas anteriores, así que no se volverán a explicar.

La función creaPiezas, crea cada una de las cuatro piezas, es decir, a cada pieza del array de piezas se le asignan los valores iniciales asociados a la pieza elegida, tales como la posición inicial (en coordenadas normalizadas de 0 a 1), el tamaño (también en coordenadas normalizadas) y la textura correspondiente. Además se pone a 0 la variable *arrastrar* ya que al principio no hay ninguna pieza siendo arrastrada.

```
static void creaPiezas() {

    piezas[0].posx = 0.6;
    piezas[0].posy = 0.7;
    piezas[0].tamx = 0.2;
    piezas[0].tamy = 0.2;
    piezas[0].arrastrar = 0;
    piezas[0].text = texturas[0];
    piezas[1].posx = 0.4;
    piezas[1].posy = 0.4;
    piezas[1].tamx = 0.2;
    piezas[1].tamy = 0.2;
    piezas[1].arrastrar = 0;
    piezas[1].text = texturas[1];
    piezas[2].posx = 0.7;
```

```
        piezas[2].posy = 0.2;
        piezas[2].tamx = 0.2;
        piezas[2].tamy = 0.2;
        piezas[2].text = texturas[2];
        piezas[2].arrastrar = 0;
        piezas[3].posx = 0.2;
        piezas[3].posy = 0.7;
        piezas[3].tamx = 0.2;
        piezas[3].tamy = 0.2;
        piezas[3].text = texturas[3];
        piezas[3].arrastrar = 0;

    }
```

La función creaPuntero, de forma parecida a *creaPiezas*, asigna al puntero sus valores iniciales de tamaño, posición, textura y máscara. La variable arrastrar tiene valor 0 puesto que el puntero no está arrastrando, todavía, ninguna pieza.

```
    static void creaPuntero() {
        puntero.posx = 0.5;
        puntero.posy = 0.5;
        puntero.tamx = 0.05;
        puntero.tamy = 0.1;
        puntero.arrastrar = 0;
        puntero.text = texturas[4];
        puntero.mask = mascaras[0];
    }
```

La función dibujaPiezas, se encarga de dibujar cada una de las piezas en la posición que le corresponda.

```
static void dibujaPiezas(int pos) {

    int i;
    float z;
    /* inicializa los valores de la vista */
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(0.0,1.0,0.0,1.0);

    glColor4f(0.0,0.0,0.0,1.0);
```

Con este bucle dibujamos todas las piezas

```
    for(i=0;i<4;i++) {
```

La variable *z* representa la profundidad, es decir, la distancia del objeto a la cámara. Si la pieza que estamos dibujando está siendo arrastrada por el puntero, se dibuja encima del resto, para ello le asignamos z=0.5 para que este más cerca de la "cámara" y a las demás les ponemos z=0 para que estén detrás de esta pieza.

```
        if (piezas[i].arrastrar) z=0.5; else z=0;
```

Se le dice a OpenGL que utilice la textura de la pieza que está siendo dibujada (*piezas[i]*)

```
        glBindTexture(GL_TEXTURE_2D,piezas[i].text.ID);
```

Las piezas se dibujan de forma parecida a como dibujabamos el rectángulo del programa de la pelotita, a partir de su centro (*posx, posy*) y conociendo el tamaïño del lado en X y en Y, sumando y restando alternativamente la mitad del tamaïño X y tamaño Y a las coordenadas X e Y del centro se obtienen las cuatro esquinas de la pieza, y por tanto los cuatro vertices del cuadrado a dibujar. Mediante glTexCoord2f se hace corresponder la figura dibujada con la textura.

```
    glBegin(GL_POLYGON);
    glTexCoord2f(1.0,1.0); glVertex3f(piezas[i].posx-piezas[i].tamx/2,piezas[i].posy-piezas[i].
   tamy/2,z);
    glTexCoord2f(1.0,0.0); glVertex3f(piezas[i].posx-piezas[i].tamx/2,piezas[i].posy+piezas[i].
   tamy/2,z);
    glTexCoord2f(0.0,0.0); glVertex3f(piezas[i].posx+piezas[i].tamx/2,piezas[i].posy+piezas[i].
   tamy/2,z);
    glTexCoord2f(0.0,1.0); glVertex3f(piezas[i].posx+piezas[i].tamx/2,piezas[i].posy-piezas[i].
   tamy/2,z);

    glEnd();
   }
   /* Vacia el buffer de dibujo */
   glFlush();
}
```

La función dibujaPuntero, es algo más complicada ya que incorpora máscaras para simular la transparencia. Las máscaras son imágenes que, aplicadas a una textura, sirven como un mapa de transparencias que indica que zonas de la textura son transparentes y cuales son más opacas. Las zonas de color negro de la máscara, aparecen opacas, mientras que las que son blancas aparecen transparentes, de este modo podemos eliminar el fondo del puntero para que no se vea que en realidad es un rectángulo. Para crear una máscara basta con editar una copia de la imagen de modo que el dibujo del puntero quede totalmente negro y todo lo demás (el fondo) quede de color blanco.

```
static void dibujaPuntero() {

 glMatrixMode(GL_PROJECTION);
 glLoadIdentity();
 gluOrtho2D(0.0, 1.0, 0.0, 1.0);
```

La líneas *glEnable(GL_BLEND)*, es necesaria para poder utilizar las máscara. Luego, mediante g*lBindTexture*, le decimos a OpenGL que utilice como textura la máscara correspondiente. Pasando la opcion *GL_ZERO* a *glBlendFunc* le estamos diciendo que las zonas negras de la textura se pinten en pantalla de color negro. La opción *GL_DST_COLOR* provoca que las zonas blancas no se pinten.

```
 glEnable(GL_BLEND);
 glBindTexture(GL_TEXTURE_2D,puntero.mask.ID);
 glBlendFunc(GL_DST_COLOR,GL_ZERO);
```

Luego se dibuja un rectángulo sobre el que se coloca la textura máscara.

```
  glBegin(GL_POLYGON);
  glTexCoord2f(1.0,1.0); glVertex3f(puntero.posx-puntero.tamx/2,
                                    puntero.posy-puntero.tamy/2,1);
   glTexCoord2f(1.0,0.0); glVertex3f(puntero.posx-puntero.tamx/2,
          puntero.posy+puntero.tamy/2,1);
   glTexCoord2f(0.0,0.0); glVertex3f(puntero.posx+puntero.tamx/2,
```

```
                                puntero.posy+puntero.tamy/2,1);
  glTexCoord2f(0.0,1.0); glVertex3f(puntero.posx+puntero.tamx/2,
                                                puntero.posy-puntero.tamy/2,1);
 glEnd();
```

Ahora le decimos que en vez de usar la máscara como textura, utilice la textura del puntero. Pasándole a *glBlendFunc* la opcion *GL_ONE* se consigue que la textura solo se pinte en aquellas zonas del rectángulo que están en negro (las que estaba en negro en la mácara), consiguiendo así que solo se dibuje la parte que nos interesa.

```
glBlendFunc(GL_ONE,GL_ONE);
glBindTexture(GL_TEXTURE_2D,puntero.text.ID);
```

Y volvemos a dibujar el rectángulo con la nueva textura.

```
glBegin(GL_POLYGON);
glTexCoord2f(1.0,1.0); glVertex3f(puntero.posx-puntero.tamx/2,puntero.posy-puntero.tamy/2,1);
glTexCoord2f(1.0,0.0); glVertex3f(puntero.posx-puntero.tamx/2,puntero.posy+puntero.tamy/2,1);
glTexCoord2f(0.0,0.0); glVertex3f(puntero.posx+puntero.tamx/2,puntero.posy+puntero.tamy/2,1);
glTexCoord2f(0.0,1.0); glVertex3f(puntero.posx+puntero.tamx/2,puntero.posy-puntero.tamy/2,1);
glEnd();
```

Se desactiva la opción *GL_BLEND* antes de finalizar la función.

```
glDisable(GL_BLEND);
}
```

La función comprobarColision, comprueba si existe una colisión entre el puntero y alguna pieza y en caso positivo devuelve el número de pieza con el que colisiona o 0 en caso contrario. La comprobación se realiza comprobando si la distancia entre los centros del puntero y de cada pieza, es menor que la suma del tamaño del lado del rectángulo del puntero y el de la pieza.

```
static int comprobarColision() {

  int i;
  for(i=0;i<4;i++) {
   if((puntero.posx+puntero.tamx/2>=piezas[i].posx-piezas[i].tamx/2)
    && (puntero.posx- puntero.tamx/2 <= piezas[i].posx+piezas[i].tamx/2)
    && (puntero.posy+puntero.tamy/2>=piezas[i].posy-piezas[i].tamy/2)
    && (puntero.posy-puntero.tamy/2 <= piezas[i].posy+piezas[i].tamy/2)){
     return i+1;
   }
  }
  return 0;
}
```

La función comprobarPiezas, tiene dos objetivos, por un lado crea un efecto imán de modo que cuando un lado de la pieza arrastrada está suficientemente cerca del lado de otra pieza, la pieza arrastrada se “pega” a la otra, facilitando así la tarea de encajar la pieza en su posición exacta. Por otro lado comprueba si se ha logrado colocar cada pieza en su sitio.

Para realizar el efecto imán, hay que recorrer todas las piezas (excepto la que se está arrastrando, *pos*) y comprobar si el lado derecho de la pieza arrastrada se acerca al lado izquierdo de otra de las piezas en menos de 0.025 unidades (en coordenadas normalizadas), en tal caso, se cambia la posición de la pieza arrastrada de modo que la posición del lado derecho de ésta coincida

exactamente con la posición del lado izquierdo de la otra. De lo contrario se comprueba si el lado izquierdo de la pieza arrastrada se acerca al lado derecho de otra pieza y se actúa de forma parecida. Luego se realiza la misma operación con los lados superior e inferior de la pieza arrastrada.

```
for(i=0;i<4;i++) {

 if(i!=pos && (piezas[pos].posx+piezas[pos].tamx/2>=piezas[i].posx-piezas[i].tamx/2-0.025) &&
(piezas[pos].posx<piezas[i].posx-piezas[i].tamx)){

   if(i!=pos && ((piezas[pos].posy>=piezas[i].posy-0.025)
      && piezas[pos].posy<piezas[i].posy) || ((piezas[i].posy>=piezas[pos].posy-0.025)
       && piezas[i].posy<piezas[pos].posy)){

          piezas[pos].posx=piezas[i].posx-piezas[i].tamx;
          piezas[pos].posy-=piezas[pos].posy-piezas[i].posy;
    }//end_if
}//end_if

 else if(i!=pos && (piezas[i].posx+piezas[i].tamx/2>=piezas[pos].posx-piezas[pos].tamx/2-0.025)
&& (piezas[i].posx<piezas[pos].posx-piezas[pos].tamx)){

  if(i!=pos && ((piezas[pos].posy>=piezas[i].posy-0.025)
     && piezas[pos].posy<piezas[i].posy) || ((piezas[i].posy>=piezas[pos].posy-0.025)
     && piezas[i].posy<piezas[pos].posy)){
          piezas[pos].posx=piezas[i].posx+piezas[i].tamx;
          piezas[pos].posy-=piezas[pos].posy-piezas[i].posy;
         }//end_if
}//end_else

 else if(i!=pos && (piezas[i].posy+piezas[i].tamy/2 >=
        piezas[pos].posy-piezas[pos].tamy/2-0.025)
         && (piezas[i].posy<piezas[pos].posy-piezas[pos].tamy)){

          if(i!=pos && ((piezas[pos].posx>=piezas[i].posx-0.025) &&
           piezas[pos].posx<piezas[i].posx) || ((piezas[i].posx>=piezas[pos].posx-0.025)
           && piezas[i].posx<piezas[pos].posx)){
              piezas[pos].posy=piezas[i].posy+piezas[i].tamy;
              piezas[pos].posx-=piezas[pos].posx-piezas[i].posx;
          }//end_if
}//end_else

 else if(i!=pos && (piezas[pos].posy+piezas[pos].tamy/2 >=
        piezas[i].posy-piezas[i].tamy/2-0.025)
        && (piezas[pos].posy<piezas[i].posy-piezas[i].tamy)){

          if(i!=pos && ((piezas[pos].posx>=piezas[i].posx-0.025) &&
           piezas[pos].posx<piezas[i].posx) || ((piezas[i].posx>=piezas[pos].posx-0.025)
           && piezas[i].posx<piezas[pos].posx)){
             piezas[pos].posy=piezas[i].posy-piezas[i].tamy;
             piezas[pos].posx-=piezas[pos].posx-piezas[i].posx;
          }//end_if

   }//end_else
}//end_for
```

Para comprobar si todas las piezas están en su posición, estableceremos un punto de referencia que

nos servirá para comprobar la posición de cada pieza respecto de ese punto. El punto de referencia elegido es la esquina inferior derecha de la primera pieza, ya que este es el único punto en que coinciden todas las piezas y podemos comprobar la posición de todas con pocas comprobaciones.

```
prefx=piezas[0].posx+piezas[0].tamx/2;
prefy=piezas[0].posy-piezas[0].tamy/2;
```

Para saber si estan bien colocadas el resto de piezas respecto de la primera, se comprueba si la esquina correspondiente a cada pieza, coincide con el punto de referencia. Si la esquina no coincide con ese punto es que no está en su posición correcta. Si todas las piezas están correctamente colocadas se devuelve un 1, de lo contrario se devuelve 0.

```
if((piezas[1].posx-piezas[1].tamx/2 == prefx && piezas[1].posy-piezas[1].tamy/2 == prefy)
    && (piezas[2].posx+piezas[2].tamx/2 == prefx
    && piezas[2].posy+piezas[2].tamy/2 == prefy)
    && (piezas[3].posx-piezas[3].tamx/2 == prefx
    && piezas[3].posy+piezas[3].tamy/2 == prefy))
{
 return 1;
}

return 0;
}
```

La función init, además de las operaciones usuales de esta función, incluye llamadas a *creaPuntero* y *creaPiezas*, para que se inicialicen el puntero y las piezas. Además se utiliza la función *cargarTGA* para cargar todas las texturas y máscaras necesarias.

```
#ifdef _WIN32
  if(!cargarTGA("images\\me1.tga", &texturas[0]) ||
    !cargarTGA("images\\me2.tga", &texturas[1]) ||
    !cargarTGA("images\\me3.tga", &texturas[2]) ||
    !cargarTGA("images\\me4.tga", &texturas[3]) ||
    !cargarTGA("images\\puntflecha.tga", &texturas[4]) ||
    !cargarTGA("images\\puntmano.tga", &texturas[5]) ||
    !cargarTGA("images\\puntcogiendo.tga", &texturas[6]) ||
    !cargarTGA("images\\puntflecha-mask.tga", &mascaras[0]) ||
    !cargarTGA("images\\puntmano-mask.tga", &mascaras[1]) ||
    !cargarTGA("images\\puntcogiendo-mask.tga", &mascaras[2])) {
#else
  if(!cargarTGA("images/me1.tga", &texturas[0]) ||
    !cargarTGA("images/me2.tga", &texturas[1]) ||
    !cargarTGA("images/me3.tga", &texturas[2]) ||
    !cargarTGA("images/me4.tga", &texturas[3]) ||
    !cargarTGA("images/puntflecha.tga", &texturas[4]) ||
   !cargarTGA("images/puntmano.tga", &texturas[5]) ||
   !cargarTGA("images/puntcogiendo.tga", &texturas[6]) ||
   !cargarTGA("images/puntflecha-mask.tga", &mascaras[0]) ||
   !cargarTGA("images/puntmano-mask.tga", &mascaras[1]) ||
   !cargarTGA("images/puntcogiendo-mask.tga", &mascaras[2])) {

#endif
    printf("Error cargando imagenes\n");
    exit(1); // Cargamos la textura y chequeamos por errores
     }
```

```
creaPuntero();
creaPiezas();
```

La función mainLoop, primero detecta los patrones de la imagen como de costumbre, dando por supuesto que solo existe un patrón y que éste se encuentra en la posición 0, por tanto el patrón será siempre *marker2[0]*. Luego se comprueba si se ha realizado un *click*, de forma identica a como haciamos en **Clicksample** ha excepción de que en esta ocasión dejamos 2 segundos para que vuelva a aparecer el patrón.

```
if(timer==0 && marker_num2==0) {
 click=0;
 timer = (long) time(NULL);
}
else if (timer>0 && marker_num2>0){
        if((time(NULL)-timer)<=2){click=1;}
 timer = 0;
}
else {
 click=0;
}
```

A continuación se invoca a *comprobarColision* y el valor devuelto (número de pieza con que colisiona o 0) se almacena una variable. Si el valor devuelto no es 0, sabemos que hay una pieza señalada por el puntero, si además se ha realizado un *click* en esta iteración tenemos que comprobar si la pieza no está siendo arrastrada ya, y si el puntero no está ocupado arrastrando alguna pieza. En caso de que se cumpla ésto, pasamos a arrastrar esta pieza, por lo que hay que poner a 1 el valor de la variable *arrastrar* tanto del puntero como de la pieza. Si en cambio la pieza ya estaba siendo arrastrada, la soltamos, dejando a 0 la variable *arrastrar* de a pieza y del puntero.

```
seleccionada = comprobarColision();
if(seleccionada){
        if(click) {
                if(!piezas[seleccionada-1].arrastrar && !puntero.arrastrar) {
                 piezas[seleccionada-1].arrastrar = 1;
                 puntero.arrastrar = 1;

                }
                else if(piezas[seleccionada-1].arrastrar) {
                 piezas[seleccionada-1].arrastrar = 0;
                 puntero.arrastrar = 0;

                }

        }//end_if

}//end_if
```

Ahora, si el puntero está arrastrando alguna pieza (su valor de *arrastrar* es 1) recorremos el array de piezas para comprobar cual de ellas está siendo arrastrada, y una vez encontrada la ponemos en la posición que viene determinada por el centro del patrón (*marker2[0].pos[0]* y *marker2[0].pos [1]*), de este modo la pieza seguirá al patrón puntero.

```
for(k=0;puntero.arrastrar && k<4;k++){
 if(piezas[k].arrastrar==1) {
    piezas[k].posx=(double)marker2[0].pos[0]/(double)arImXsize;
    piezas[k].posy=1.0-((double)marker2[0].pos[1]/(double)arImYsize);
 }
}
```

Se comprueba si se ha encontrado el patrón puntero, y si es el caso, la posición del puntero que hay que dibujar se corresponderá con el centro del patrón.

```
if(marker_num2>0) {
 puntero.posx=marker2[0].pos[0]/(double)arImXsize;
 puntero.posy=1-marker2[0].pos[1]/(double)arImYsize;
}
```

Hay que asignar diferentes texturas (y la máscara correspondiente según la textura) al puntero según sea su estado: arrastrando, colisionando con una pieza o estado normal.

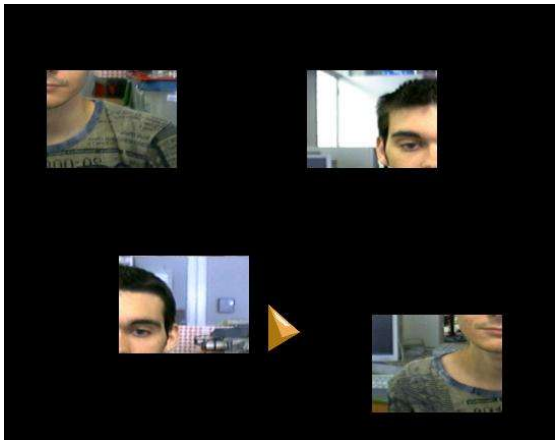
```
if(puntero.arrastrar) {
        puntero.text = texturas[6]; puntero.mask = mascaras[2];
}
else if(seleccionada){
        puntero.text = texturas[5]; puntero.mask = mascaras[1];
}
else {
        puntero.text = texturas[4]; puntero.mask = mascaras[0];
                }
```

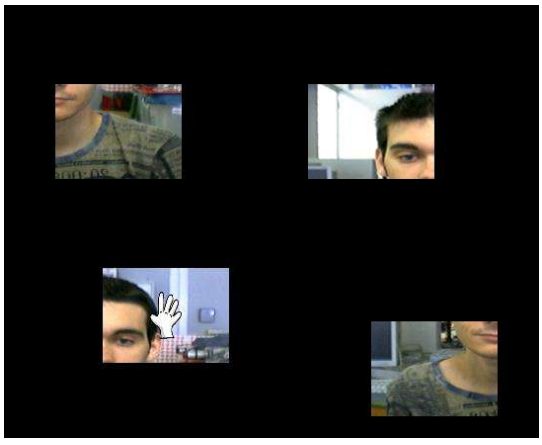
Ahora se comprueba la posición de las piezas y si es correcta se muestra un mensaje.

```
if (comprobarPiezas(seleccionada-1)) {
 printf("Correcto!");
}
```

Por último se dibujan las piezas y el puntero en sus nuevas posiciones. A *dibujaPiezas* se le pasa el número de pieza seleccionada para que sepa que pieza tiene que dibujar encima de todas las demás.

```
  dibujaPiezas(seleccionada-1);
  dibujaPuntero();
  argSwapBuffers();
}
```





6. Conclusiones

Hemos añadido funciones a las librerías de modo que pueda utilizarse patrones de cualquier forma y color, que son más naturales y fáciles de conseguir que los originales.

Se ha elaborado una pequeña utilidad de configuración que ayuda a seleccionar un patrón de un color determinado guardando la información sobre el patrón en un fichero, al tiempo que sirve como demostración de como se pueden utilizar las nuevas funciones de las librerías para detectar patrones.

También hemos creado algunos ejemplos de uso de las nuevas funciones, que muestran como puede utilizarse uno de estos patrones para simular un ratón o joystick sencillos.

Se ha mostrado también, mediante un ejemplo, la forma de usar varios patrones de distinto color en la misma imagen, siendo uno de ellos un patrón puntero y los demás patrones objetivo.

Se han creado versiones en código fuente y binarias de los programas y de las librerías. Además, se ha elaborado la documentación a dos niveles: para un usuario común y para un usuario desarrollador. La finalidad de esta documentación es servir como guía de uso para el usuario común, y al mismo tiempo procura dar la información necesaria al usuario desarrollador para que pueda entender las modificaciones realizadas y realizar sus propios programas, o modificaciones, sobre esta versión de ARToolKit.

Durante el desarrollo de este proyecto, se ha intentado conseguir que las librerías y programas sean multiplataforma, pudiendo ejecutarse tanto en sistemas operativos Windows como Linux. Esto se ha conseguido mediante el uso de ARToolKit, OpenGL y Allegro, todos ellos con versiones para ambos sistemas.

7. Trabajos futuros

Se citan a continuación algunos de las posibles mejoras que podrían realizarse sobre esta versión de ARToolKit modificada:

- Ejemplos que muestren como pueden combinarse los patrones de colores, con los patrones propios de ARToolKit combinando las ventajas de cada uno de ellos.
- Mejoras en la utilidad de configuración, solucionando algún error y proporcionando métodos más fáciles, rápidos y efectivos de elegir el patrón a utilizar.
- Posibilidad de guardar cada patrón en un fichero diferente en lugar de tener únicamente el

fichero conf.con.
- Interfaces gráficas para los ejemplos y utilidades que ofrezcan al usuario mayor comodidad y facilidad de uso.
- Mejoras en la detección de colores en la imagen, así como uso de algoritmos más eficientes.
- Ejemplos más sofisticados: juegos más realistas y divertidos, aplicaciones útiles, etc.
- Funciones que detecten no solo el color, sino que puedan detectar también forma, aspecto, orientación y movimiento, tanto en 2 dimensiones como en 3. Esto permitiría utilizar patrones más naturales (mano, ojos, etc.) y obtener más información de ellos.

8. Bibliografía

Libros

- *Visión por computador: fundamentos y métodos*. Escalera Hueso, Arturo de la

Direcciones Web

- *ARToolKit*. Human Interface Technology Laboratory. http://www.hitl.washington.edu/artoolkit
- *Visión por computador*. Domingo Mery. http://www2.ing.puc.cl/~dmery/vision/visionc.pdf
- *La Realidad Virtual*. Javier B. Galeano G. www.monografias.com/trabajos4/realvirtual/realvirtual.shtml
- *OpenGL*. The Industry's Foundation for High Performance Graphics. http://www.opengl.org
- *OpenGL+Glut*. Correa Villanueva, Javier. http://articulos.conclase.net/glut/glut004.html
- *Manual de Allegro*. Shawn Hargreaves. http://es.tldp.org/Allegro-es/web/online/allegro.html

ANEXOS

Fichero arDetectMarker2.c

```
#include <AR/ar.h>

static int check_square( int area, ARMarkerInfo2 *marker_info2, double factor );

static int get_vertex( int x_coord[], int y_coord[], int st, int ed,
                       double thresh, int vertex[], int *vnum );

static ARMarkerInfo2    marker_info2[AR_SQUARE_MAX];

ARMarkerInfo2 *arDetectMarker2( ARInt16 *limage, int label_num, int *label_ref,
                            int *warea, double *wpos, int *wclip,
                            int area_max, int area_min, double factor, int *marker_num )
{
   ARMarkerInfo2    *pm;
   int              xsize, ysize;
   int              marker_num2;
   int              i, j, ret;
   double           d;

   if( arImageProcMode == AR_IMAGE_PROC_IN_HALF ) {
      area_min /= 4;
      area_max /= 4;
      xsize = arImXsize / 2;
      ysize = arImYsize / 2;
   }
   else {
      xsize = arImXsize;
      ysize = arImYsize;
   }
   marker_num2 = 0;
   for(i=0; i<label_num; i++ ) {
      if( warea[i] < area_min || warea[i] > area_max ) continue;
      if( wclip[i*4+0] == 1 || wclip[i*4+1] == xsize-2 ) continue;
      if( wclip[i*4+2] == 1 || wclip[i*4+3] == ysize-2 ) continue;

      ret = arGetContour( limage, label_ref, i+1,
                          &(wclip[i*4]), &(marker_info2[marker_num2]));
      if( ret < 0 ) continue;

      ret = check_square( warea[i], &(marker_info2[marker_num2]), factor );
      if( ret < 0 ) continue;

      marker_info2[marker_num2].area   = warea[i];
      marker_info2[marker_num2].pos[0] = wpos[i*2+0];
      marker_info2[marker_num2].pos[1] = wpos[i*2+1];
      marker_num2++;
      if( marker_num2 == AR_SQUARE_MAX ) break;
   }
```

```
    for( i=0; i < marker_num2; i++ ) {
        for( j=i+1; j < marker_num2; j++ ) {
            d = (marker_info2[i].pos[0] - marker_info2[j].pos[0])
              * (marker_info2[i].pos[0] - marker_info2[j].pos[0])
              + (marker_info2[i].pos[1] - marker_info2[j].pos[1])
              * (marker_info2[i].pos[1] - marker_info2[j].pos[1]);
            if( marker_info2[i].area > marker_info2[j].area ) {
                if( d < marker_info2[i].area / 4 ) {
                    marker_info2[j].area = 0;
                }
            }
            else {
                if( d < marker_info2[j].area / 4 ) {
                    marker_info2[i].area = 0;
                }
            }
        }
    }
    for( i=0; i < marker_num2; i++ ) {
        if( marker_info2[i].area == 0.0 ) {
            for( j=i+1; j < marker_num2; j++ ) {
                marker_info2[j-1] = marker_info2[j];
            }
            marker_num2--;
        }
    }

    if( arImageProcMode == AR_IMAGE_PROC_IN_HALF ) {
        pm = &(marker_info2[0]);
        for( i = 0; i < marker_num2; i++ ) {
            pm->area *= 4;
            pm->pos[0] *= 2.0;
            pm->pos[1] *= 2.0;
            for( j = 0; j< pm->coord_num; j++ ) {
                pm->x_coord[j] *= 2;
                pm->y_coord[j] *= 2;
            }
            pm++;
        }
    }

    *marker_num = marker_num2;
    return( &(marker_info2[0]) );
}

int arGetContour( ARInt16 *limage, int *label_ref,
                  int label, int clip[4], ARMarkerInfo2 *marker_info2 )
{
    static int      xdir[8] = { 0, 1, 1, 1, 0,-1,-1,-1};
    static int      ydir[8] = {-1,-1, 0, 1, 1, 1, 0,-1};
    static int      wx[AR_CHAIN_MAX];
    static int      wy[AR_CHAIN_MAX];
    ARInt16         *p1;
```

```
int             xsize, ysize;
int             sx, sy, dir;
int             dmax, d, v1;
int             i, j;

if( arImageProcMode == AR_IMAGE_PROC_IN_HALF ) {
    xsize = arImXsize / 2;
    ysize = arImYsize / 2;
}
else {
    xsize = arImXsize;
    ysize = arImYsize;
}
j = clip[2];
p1 = &(limage[j*xsize+clip[0]]);
for( i = clip[0]; i <= clip[1]; i++, p1++ ) {
    if( *p1 > 0 && label_ref[(*p1)-1] == label ) {
        sx = i; sy = j; break;
    }
}
if( i > clip[1] ) {
    printf("??? 1\n"); return(-1);
}

marker_info2->coord_num = 1;
marker_info2->x_coord[0] = sx;
marker_info2->y_coord[0] = sy;
dir = 5;
for(;;) {
    p1 = &(limage[marker_info2->y_coord[marker_info2->coord_num-1] * xsize
                + marker_info2->x_coord[marker_info2->coord_num-1]]);
    dir = (dir+5)%8;
    for(i=0;i<8;i++) {
        if( p1[ydir[dir]*xsize+xdir[dir]] > 0 ) break;
        dir = (dir+1)%8;
    }
    if( i == 8 ) {
        printf("??? 2\n"); return(-1);
    }
    marker_info2->x_coord[marker_info2->coord_num]
        = marker_info2->x_coord[marker_info2->coord_num-1] + xdir[dir];
    marker_info2->y_coord[marker_info2->coord_num]
        = marker_info2->y_coord[marker_info2->coord_num-1] + ydir[dir];
    if( marker_info2->x_coord[marker_info2->coord_num] == sx
     && marker_info2->y_coord[marker_info2->coord_num] == sy ) break;
    marker_info2->coord_num++;
    if( marker_info2->coord_num == AR_CHAIN_MAX-1 ) {
        printf("??? 3\n"); return(-1);
    }
}

dmax = 0;
for(i=1;i<marker_info2->coord_num;i++) {
```

```
        d = (marker_info2->x_coord[i]-sx)*(marker_info2->x_coord[i]-sx)
          + (marker_info2->y_coord[i]-sy)*(marker_info2->y_coord[i]-sy);
        if( d > dmax ) {
            dmax = d;
            v1 = i;
        }
    }

    for(i=0;i<v1;i++) {
        wx[i] = marker_info2->x_coord[i];
        wy[i] = marker_info2->y_coord[i];
    }
    for(i=v1;i<marker_info2->coord_num;i++) {
        marker_info2->x_coord[i-v1] = marker_info2->x_coord[i];
        marker_info2->y_coord[i-v1] = marker_info2->y_coord[i];
    }
    for(i=0;i<v1;i++) {
        marker_info2->x_coord[i-v1+marker_info2->coord_num] = wx[i];
        marker_info2->y_coord[i-v1+marker_info2->coord_num] = wy[i];
    }
    marker_info2->x_coord[marker_info2->coord_num] = marker_info2->x_coord[0];
    marker_info2->y_coord[marker_info2->coord_num] = marker_info2->y_coord[0];
    marker_info2->coord_num++;

    return 0;
}

static int check_square( int area, ARMarkerInfo2 *marker_info2, double factor )
{
    int         sx, sy;
    int         dmax, d, v1;
    int         vertex[10], vnum;
    int         wv1[10], wvnum1, wv2[10], wvnum2, v2;
    double      thresh;
    int         i;


    dmax = 0;
    v1 = 0;
    sx = marker_info2->x_coord[0];
    sy = marker_info2->y_coord[0];
    for(i=1;i<marker_info2->coord_num-1;i++) {
        d = (marker_info2->x_coord[i]-sx)*(marker_info2->x_coord[i]-sx)
          + (marker_info2->y_coord[i]-sy)*(marker_info2->y_coord[i]-sy);
        if( d > dmax ) {
            dmax = d;
            v1 = i;
        }
    }

    thresh = (area/0.75) * 0.01 * factor;
    vnum = 1;
    vertex[0] = 0;
```

```
wvnum1 = 0;
wvnum2 = 0;
if( get_vertex(marker_info2->x_coord, marker_info2->y_coord, 0,  v1,
            thresh, wv1, &wvnum1) < 0 ) {
   return(-1);
}
if( get_vertex(marker_info2->x_coord, marker_info2->y_coord,
            v1,  marker_info2->coord_num-1, thresh, wv2, &wvnum2) < 0 ) {
   return(-1);
}

if( wvnum1 == 1 && wvnum2 == 1 ) {
   vertex[1] = wv1[0];
   vertex[2] = v1;
   vertex[3] = wv2[0];
}
else if( wvnum1 > 1 && wvnum2 == 0 ) {
   v2 = v1 / 2;
   wvnum1 = wvnum2 = 0;
   if( get_vertex(marker_info2->x_coord, marker_info2->y_coord,
               0,  v2, thresh, wv1, &wvnum1) < 0 ) {
      return(-1);
   }
   if( get_vertex(marker_info2->x_coord, marker_info2->y_coord,
               v2,  v1, thresh, wv2, &wvnum2) < 0 ) {
      return(-1);
   }
   if( wvnum1 == 1 && wvnum2 == 1 ) {
      vertex[1] = wv1[0];
      vertex[2] = wv2[0];
      vertex[3] = v1;
   }
   else {
      return(-1);
   }
}
else if( wvnum1 == 0 && wvnum2 > 1 ) {
   v2 = (v1 + marker_info2->coord_num-1) / 2;
   wvnum1 = wvnum2 = 0;
   if( get_vertex(marker_info2->x_coord, marker_info2->y_coord,
            v1, v2, thresh, wv1, &wvnum1) < 0 ) {
      return(-1);
   }
   if( get_vertex(marker_info2->x_coord, marker_info2->y_coord,
            v2, marker_info2->coord_num-1, thresh, wv2, &wvnum2) < 0 ) {
      return(-1);
   }
   if( wvnum1 == 1 && wvnum2 == 1 ) {
      vertex[1] = v1;
      vertex[2] = wv1[0];
      vertex[3] = wv2[0];
   }
   else {
```

```
            return(-1);
        }
    }
    else {
        return(-1);
    }

    marker_info2->vertex[0] = vertex[0];
    marker_info2->vertex[1] = vertex[1];
    marker_info2->vertex[2] = vertex[2];
    marker_info2->vertex[3] = vertex[3];
    marker_info2->vertex[4] = marker_info2->coord_num-1;

    return(0);
}

static int get_vertex( int x_coord[], int y_coord[], int st,  int ed,
                       double thresh, int vertex[], int *vnum)
{
    double   d, dmax;
    double   a, b, c;
    int      i, v1;

    a = y_coord[ed] - y_coord[st];
    b = x_coord[st] - x_coord[ed];
    c = x_coord[ed]*y_coord[st] - y_coord[ed]*x_coord[st];
    dmax = 0;
    for(i=st+1;i<ed;i++) {
        d = a*x_coord[i] + b*y_coord[i] + c;
        if( d*d > dmax ) {
            dmax = d*d;
            v1 = i;
        }
    }
    if( dmax/(a*a+b*b) > thresh ) {
        if( get_vertex(x_coord, y_coord, st,  v1, thresh, vertex, vnum) < 0 )
            return(-1);

        if( (*vnum) > 5 ) return(-1);
        vertex[(*vnum)] = v1;
        (*vnum)++;

        if( get_vertex(x_coord, y_coord, v1,  ed, thresh, vertex, vnum) < 0 )
            return(-1);
    }

    return(0);
}


ARMarkerInfo2 *arDetectMarker3( ARInt16 *limage, int label_num, int *label_ref,
                                int *warea, double *wpos, int *wclip,
                                                          int area_max, int area_min, int
```

```
*marker_num )
{
   ARMarkerInfo2    *pm;
   int              xsize, ysize;
   int              marker_num2;
   int              i, j, ret;
   double           d;

   if( arImageProcMode == AR_IMAGE_PROC_IN_HALF ) {
      area_min /= 4;
      area_max /= 4;
      xsize = arImXsize / 2;
      ysize = arImYsize / 2;
   }
   else {
      xsize = arImXsize;
      ysize = arImYsize;
   }
   marker_num2 = 0;
   for(i=0; i<label_num; i++ ) {
      if( warea[i] < area_min || warea[i] > area_max ) continue;   //si el area es mayor o menor que
los valores maximos y minimos, nos la saltamos
      if( wclip[i*4+0] == 1 || wclip[i*4+1] == xsize-2 ) continue;  //Si ocupa toda la pantalla nos la
saltamos..?
      if( wclip[i*4+2] == 1 || wclip[i*4+3] == ysize-2 ) continue;  //Si ocupa toda la pantalla nos la
saltamos..?

      ret = arGetContour( limage, label_ref, i+1,
                     &(wclip[i*4]), &(marker_info2[marker_num2]));
      if( ret < 0 ) continue;  //si ha habido error nos saltamos esta etiqueta y continuamos con la
siguiente iteración

      //ret = check_square( warea[i], &(marker_info2[marker_num2]), factor );
      //if( ret < 0 ) continue;

      marker_info2[marker_num2].area   = warea[i];
      marker_info2[marker_num2].pos[0] = wpos[i*2+0];
      marker_info2[marker_num2].pos[1] = wpos[i*2+1];
      marker_num2++;
      //if( marker_num2 == AR_SQUARE_MAX ) break;
   }

   for( i=0; i < marker_num2; i++ ) {                          //para cada marca se calculan que
marcas son externas y cuales internas a otras marcas
      for( j=i+1; j < marker_num2; j++ ) {                      //las internas no nos valen
         d = (marker_info2[i].pos[0] - marker_info2[j].pos[0])          // d = distancia entre una
marca y la siguiente
           * (marker_info2[i].pos[0] - marker_info2[j].pos[0])           //pos[0] es la coord X de la
posicion de la marca
           + (marker_info2[i].pos[1] - marker_info2[j].pos[1])           //pos[1] es la coord Y de la
posicion de la marca
           * (marker_info2[i].pos[1] - marker_info2[j].pos[1]);
         if( marker_info2[i].area > marker_info2[j].area ) {    //si la marca i es mas grande que la j
```

```
            if( d < marker_info2[i].area / 4 ) {                //si la marca j esta dentro de la marca i
                marker_info2[j].area = 0;                        //j no es una marca --> area de j = 0
            }
        }
        else {                                          //sino si la marca j es mas grande
            if( d < marker_info2[j].area / 4 ) {                //si la marca j esta dentro de la marca i
                marker_info2[i].area = 0;                       //i no es una marca ...> area de i = 0
            }
        }
    }
  }
  for( i=0; i < marker_num2; i++ ) {                        //se eliminan de la lista de marcas
detectadas, aquellas que no son validas (internas a otras)
    if( marker_info2[i].area == 0.0 ) {
      for( j=i+1; j < marker_num2; j++ ) {
        marker_info2[j-1] = marker_info2[j];
      }
      marker_num2--;
    }
  }

  if( arImageProcMode == AR_IMAGE_PROC_IN_HALF ) {
    pm = &(marker_info2[0]);
    for( i = 0; i < marker_num2; i++ ) {
      pm->area *= 4;
      pm->pos[0] *= 2.0;
      pm->pos[1] *= 2.0;
      for( j = 0; j< pm->coord_num; j++ ) {
        pm->x_coord[j] *= 2;
        pm->y_coord[j] *= 2;
      }
      pm++;
    }
  }

  *marker_num = marker_num2;
  return( &(marker_info2[0]) );
}
```

Fichero arLabeling.c

```
#include <stdlib.h>
#include <stdio.h>
#include <AR/ar.h>
#include <formatopixel.h>

#ifdef _WIN32
#  include <windows.h>
#  define put_zero(p,s) ZeroMemory(p, s)
#else
#  include <string.h>
#  define put_zero(p,s) memset((void *)p, 0, s)
#endif

#define USE_OPTIMIZATIONS
#define WORK_SIZE   1024*32



/*******************************************************************************/
// BUG in ARToolkit 2.65
// Hardcoded buffer (600*500) is too small for full-size DV-PAL/NTSC resolutions of
// 720x576 and 720x480, respectively. Results in segment faults.
/*
static ARInt16     l_imageL[640*500];
static ARInt16     l_imageR[640*500];
*/

#define HARDCODED_BUFFER_WIDTH  1024
#define HARDCODED_BUFFER_HEIGHT 1024

static ARInt16     l_imageL
[HARDCODED_BUFFER_WIDTH*HARDCODED_BUFFER_HEIGHT];
static ARInt16     l_imageR
[HARDCODED_BUFFER_WIDTH*HARDCODED_BUFFER_HEIGHT];
/*******************************************************************************/


static int         workL[WORK_SIZE];
static int         workR[WORK_SIZE];
static int         work2L[WORK_SIZE*7];
static int         work2R[WORK_SIZE*7];

static int         wlabel_numL;
static int         wlabel_numR;
static int         wareaL[WORK_SIZE];
static int         wareaR[WORK_SIZE];
static int         wclipL[WORK_SIZE*4];
static int         wclipR[WORK_SIZE*4];
static double      wposL[WORK_SIZE*2];
static double      wposR[WORK_SIZE*2];
```

```
static ARInt16 *labeling2( ARUint8 *image, int thresh,
                    int *label_num, int **area, double **pos, int **clip,
                    int **label_ref, int LorR );
static ARInt16 *labeling3( ARUint8 *image, int thresh,
                    int *label_num, int **area, double **pos, int **clip,
                    int **label_ref, int LorR );

void arGetImgFeature( int *num, int **area, int **clip, double **pos )
{
   *num  = wlabel_numL;
   *area = wareaL;
   *clip = wclipL;
   *pos  = wposL;

   return;
}

ARInt16 *arLabeling( ARUint8 *image, int thresh,
                int *label_num, int **area, double **pos, int **clip,
                int **label_ref )
{
   if( arDebug ) {
      return( labeling3(image, thresh, label_num,
                  area, pos, clip, label_ref, 1) );
   }
   else {
      return( labeling2(image, thresh, label_num,
                  area, pos, clip, label_ref, 1) );
   }
}

void arsGetImgFeature( int *num, int **area, int **clip, double **pos, int LorR )
{
   if( LorR ) {
      *num  = wlabel_numL;
      *area = wareaL;
      *clip = wclipL;
      *pos  = wposL;
   }
   else {
      *num  = wlabel_numR;
      *area = wareaR;
      *clip = wclipR;
      *pos  = wposR;
   }

   return;
}

ARInt16 *arsLabeling( ARUint8 *image, int thresh,
                int *label_num, int **area, double **pos, int **clip,
                int **label_ref, int LorR )
{
```

```
    if( arDebug ) {
        return( labeling3(image, thresh, label_num,
                        area, pos, clip, label_ref, LorR) );
    }
    else {
        return( labeling2(image, thresh, label_num,
                        area, pos, clip, label_ref, LorR) );
    }
}

static ARInt16 *labeling2( ARUint8 *image, int thresh,
                        int *label_num, int **area, double **pos, int **clip,
                        int **label_ref, int LorR )
{
    ARUint8   *pnt;                 /*  image pointer       */
    ARInt16   *pnt1, *pnt2;         /*  image pointer       */
    int       *wk;                  /*  pointer for work    */
    int       wk_max;               /*  work                */
    int       m,n;                  /*  work                */
    int       i,j,k;                /*  for loop            */
    int       lxsize, lysize;
    int       poff;
    ARInt16   *l_image;
    int       *work, *work2;
    int       *wlabel_num;
    int       *warea;
    int       *wclip;
    double    *wpos;
#ifdef USE_OPTIMIZATIONS
        int                 pnt2_index;   // [tp]
#endif

        if( LorR ) {
        l_image = &l_imageL[0];
        work    = &workL[0];
        work2   = &work2L[0];
        wlabel_num = &wlabel_numL;
        warea   = &wareaL[0];
        wclip   = &wclipL[0];
        wpos    = &wposL[0];
    }
    else {
        l_image = &l_imageR[0];
        work    = &workR[0];
        work2   = &work2R[0];
        wlabel_num = &wlabel_numR;
        warea   = &wareaR[0];
        wclip   = &wclipR[0];
        wpos    = &wposR[0];
    }

    thresh *= 3;
    if( arImageProcMode == AR_IMAGE_PROC_IN_HALF ) {
```

```
        lxsize = arImXsize / 2;
        lysize = arImYsize / 2;
    }
    else {
        lxsize = arImXsize;
        lysize = arImYsize;
    }

    pnt1 = &l_image[0];
    pnt2 = &l_image[(lysize-1)*lxsize];

#ifndef USE_OPTIMIZATIONS
            for(i = 0; i < lxsize; i++) {
        *(pnt1++) = *(pnt2++) = 0;
    }
#else
// 4x loop unrolling
            for(i = 0; i < lxsize-(lxsize%4); i+=4) {
        *(pnt1++) = *(pnt2++) = 0;
        *(pnt1++) = *(pnt2++) = 0;
        *(pnt1++) = *(pnt2++) = 0;
        *(pnt1++) = *(pnt2++) = 0;
    }
#endif
    pnt1 = &l_image[0];
    pnt2 = &l_image[lxsize-1];

#ifndef USE_OPTIMIZATIONS
    for(i = 0; i < lysize; i++) {
        *pnt1 = *pnt2 = 0;
        pnt1 += lxsize;
        pnt2 += lxsize;
    }
#else
// 4x loop unrolling
    for(i = 0; i < lysize-(lysize%4); i+=4) {
                    *pnt1 = *pnt2 = 0;
        pnt1 += lxsize;
        pnt2 += lxsize;

                    *pnt1 = *pnt2 = 0;
        pnt1 += lxsize;
        pnt2 += lxsize;

                    *pnt1 = *pnt2 = 0;
        pnt1 += lxsize;
        pnt2 += lxsize;

                    *pnt1 = *pnt2 = 0;
        pnt1 += lxsize;
        pnt2 += lxsize;
    }
#endif
```

```
    wk_max = 0;
    pnt2 = &(l_image[lxsize+1]);
    if( arImageProcMode == AR_IMAGE_PROC_IN_HALF ) {
        pnt = &(image[(arImXsize*2+2)*AR_PIX_SIZE]);
        poff = AR_PIX_SIZE*2;
    }
    else {
        pnt = &(image[(arImXsize+1)*AR_PIX_SIZE]);
        poff = AR_PIX_SIZE;
    }
    for(j = 1; j < lysize-1; j++, pnt+=poff*2, pnt2+=2) {
        for(i = 1; i < lxsize-1; i++, pnt+=poff, pnt2++) {
#if defined(AR_PIX_FORMAT_ARGB)
            if( *(pnt+1) + *(pnt+2) + *(pnt+3) <= thresh ) {
#elif defined(AR_PIX_FORMAT_ABGR)
            if( *(pnt+1) + *(pnt+2) + *(pnt+3) <= thresh ) {
#elif defined(AR_PIX_FORMAT_BGRA)
            if( *(pnt+0) + *(pnt+1) + *(pnt+2) <= thresh ) {
#elif defined(AR_PIX_FORMAT_BGR)
            if( *(pnt+0) + *(pnt+1) + *(pnt+2) <= thresh ) {
#elif defined(AR_PIX_FORMAT_RGBA)
            if( *(pnt+0) + *(pnt+1) + *(pnt+2) <= thresh ) {
#elif defined(AR_PIX_FORMAT_RGB)
            if( *(pnt+0) + *(pnt+1) + *(pnt+2) <= thresh ) {
#elif defined(AR_PIX_FORMAT_2vuy)
                            if( *(pnt+1) * 3 <= thresh ) {
#elif defined(AR_PIX_FORMAT_yuvs)
                            if( *(pnt+0) * 3 <= thresh ) {
#else
#  error Unknown pixel format defined in config.h
#endif
                pnt1 = &(pnt2[-lxsize]);
                if( *pnt1 > 0 ) {
                    *pnt2 = *pnt1;

#ifndef USE_OPTIMIZATIONS
                                            // ORIGINAL CODE
                                            work2[((*pnt2)-1)*7+0] ++;
                    work2[((*pnt2)-1)*7+1] += i;
                    work2[((*pnt2)-1)*7+2] += j;
                    work2[((*pnt2)-1)*7+6] = j;
#else
                                            // OPTIMIZED CODE [tp]
                                            // ((*pnt2)-1)*7 should be treated as constant, since
                                            //  work2[n] (n=0..xsize*ysize) cannot overwrite (*pnt2)
                                            pnt2_index = ((*pnt2)-1) * 7;
                    work2[pnt2_index+0]++;
                    work2[pnt2_index+1]+= i;
                    work2[pnt2_index+2]+= j;
                    work2[pnt2_index+6] = j;
                                            // --------------------------------
#endif
```

```
            }
            else if( *(pnt1+1) > 0 ) {
               if( *(pnt1-1) > 0 ) {
                  m = work[*(pnt1+1)-1];
                  n = work[*(pnt1-1)-1];
                  if( m > n ) {
                     *pnt2 = n;
                     wk = &(work[0]);
                     for(k = 0; k < wk_max; k++) {
                        if( *wk == m ) *wk = n;
                        wk++;
                     }
                  }
                  else if( m < n ) {
                     *pnt2 = m;
                     wk = &(work[0]);
                     for(k = 0; k < wk_max; k++) {
                        if( *wk == n ) *wk = m;
                        wk++;
                     }
                  }
                  else *pnt2 = m;

#ifndef USE_OPTIMIZATIONS
                                                   // ORIGINAL CODE
                                                   work2[((*pnt2)-1)*7+0] ++;
                  work2[((*pnt2)-1)*7+1] += i;
                  work2[((*pnt2)-1)*7+2] += j;
                  work2[((*pnt2)-1)*7+6] = j;
#else
                                                   // PERFORMANCE OPTIMIZATION:
                                                   pnt2_index = ((*pnt2)-1) * 7;
                                                   work2[pnt2_index+0]++;
                                                   work2[pnt2_index+1]+= i;
                                                   work2[pnt2_index+2]+= j;
                                                   work2[pnt2_index+6] = j;
#endif

               }
               else if( *(pnt2-1) > 0 ) {
                  m = work[*(pnt1+1)-1];
                  n = work[*(pnt2-1)-1];
                  if( m > n ) {
                     *pnt2 = n;
                     wk = &(work[0]);
                     for(k = 0; k < wk_max; k++) {
                        if( *wk == m ) *wk = n;
                        wk++;
                     }
                  }
                  else if( m < n ) {
                     *pnt2 = m;
                     wk = &(work[0]);
```

```
                    for(k = 0; k < wk_max; k++) {
                        if( *wk == n ) *wk = m;
                        wk++;
                    }
                }
                else *pnt2 = m;

#ifndef USE_OPTIMIZATIONS
                                                // ORIGINAL CODE
                work2[((*pnt2)-1)*7+0] ++;
                work2[((*pnt2)-1)*7+1] += i;
                work2[((*pnt2)-1)*7+2] += j;
#else
                                                // PERFORMANCE OPTIMIZATION:
                                                pnt2_index = ((*pnt2)-1) * 7;
                                                work2[pnt2_index+0]++;
                                                work2[pnt2_index+1]+= i;
                                                work2[pnt2_index+2]+= j;
#endif

            }
            else {
                *pnt2 = *(pnt1+1);

#ifndef USE_OPTIMIZATIONS
                                                // ORIGINAL CODE
                work2[((*pnt2)-1)*7+0] ++;
                work2[((*pnt2)-1)*7+1] += i;
                work2[((*pnt2)-1)*7+2] += j;
                if( work2[((*pnt2)-1)*7+3] > i ) work2[((*pnt2)-1)*7+3] = i;
                work2[((*pnt2)-1)*7+6] = j;
#else
                                                // PERFORMANCE OPTIMIZATION:
                                                pnt2_index = ((*pnt2)-1) * 7;
                                                work2[pnt2_index+0]++;
                                                work2[pnt2_index+1]+= i;
                                                work2[pnt2_index+2]+= j;
                if( work2[pnt2_index+3] > i ) work2[pnt2_index+3] = i;
                                                work2[pnt2_index+6] = j;
#endif
            }
        }
        else if( *(pnt1-1) > 0 ) {
            *pnt2 = *(pnt1-1);

#ifndef USE_OPTIMIZATIONS
                                                // ORIGINAL CODE
            work2[((*pnt2)-1)*7+0] ++;
            work2[((*pnt2)-1)*7+1] += i;
            work2[((*pnt2)-1)*7+2] += j;
            if( work2[((*pnt2)-1)*7+4] < i ) work2[((*pnt2)-1)*7+4] = i;
            work2[((*pnt2)-1)*7+6] = j;
#else
```

```
                                        // PERFORMANCE OPTIMIZATION:
                                        pnt2_index = ((*pnt2)-1) * 7;
                                        work2[pnt2_index+0]++;
                                        work2[pnt2_index+1]+= i;
                                        work2[pnt2_index+2]+= j;
                if( work2[pnt2_index+4] < i ) work2[pnt2_index+4] = i;
                                        work2[pnt2_index+6] = j;
#endif
            }
            else if( *(pnt2-1) > 0) {
                *pnt2 = *(pnt2-1);

#ifndef USE_OPTIMIZATIONS
                                                // ORIGINAL CODE
                work2[((*pnt2)-1)*7+0] ++;
                work2[((*pnt2)-1)*7+1] += i;
                work2[((*pnt2)-1)*7+2] += j;
                if( work2[((*pnt2)-1)*7+4] < i ) work2[((*pnt2)-1)*7+4] = i;
#else
                                        // PERFORMANCE OPTIMIZATION:
                                        pnt2_index = ((*pnt2)-1) * 7;
                                        work2[pnt2_index+0]++;
                                        work2[pnt2_index+1]+= i;
                                        work2[pnt2_index+2]+= j;
                if( work2[pnt2_index+4] < i ) work2[pnt2_index+4] = i;
#endif
                                    }
            else {
                wk_max++;
                if( wk_max > WORK_SIZE ) {
                    return(0);
                }
                work[wk_max-1] = *pnt2 = wk_max;
                work2[(wk_max-1)*7+0] = 1;
                work2[(wk_max-1)*7+1] = i;
                work2[(wk_max-1)*7+2] = j;
                work2[(wk_max-1)*7+3] = i;
                work2[(wk_max-1)*7+4] = i;
                work2[(wk_max-1)*7+5] = j;
                work2[(wk_max-1)*7+6] = j;
            }
        }
        else {
            *pnt2 = 0;
        }
    }
    if( arImageProcMode == AR_IMAGE_PROC_IN_HALF ) pnt +=
arImXsize*AR_PIX_SIZE;
  }

  j = 1;
  wk = &(work[0]);
  for(i = 1; i <= wk_max; i++, wk++) {
```

```
        *wk = (*wk==i)? j++: work[(*wk)-1];
    }
    *label_num = *wlabel_num = j - 1;
    if( *label_num == 0 ) {
        return( l_image );
    }

    put_zero( (ARUint8 *)warea, *label_num *     sizeof(int) );
    put_zero( (ARUint8 *)wpos,  *label_num * 2 * sizeof(double) );
    for(i = 0; i < *label_num; i++) {
        wclip[i*4+0] = lxsize;
        wclip[i*4+1] = 0;
        wclip[i*4+2] = lysize;
        wclip[i*4+3] = 0;
    }
    for(i = 0; i < wk_max; i++) {
        j = work[i] - 1;
        warea[j]    += work2[i*7+0];
        wpos[j*2+0] += work2[i*7+1];
        wpos[j*2+1] += work2[i*7+2];
        if( wclip[j*4+0] > work2[i*7+3] ) wclip[j*4+0] = work2[i*7+3];
        if( wclip[j*4+1] < work2[i*7+4] ) wclip[j*4+1] = work2[i*7+4];
        if( wclip[j*4+2] > work2[i*7+5] ) wclip[j*4+2] = work2[i*7+5];
        if( wclip[j*4+3] < work2[i*7+6] ) wclip[j*4+3] = work2[i*7+6];
    }

    for( i = 0; i < *label_num; i++ ) {
        wpos[i*2+0] /= warea[i];
        wpos[i*2+1] /= warea[i];
    }

    *label_ref = work;
    *area      = warea;
    *pos       = wpos;
    *clip      = wclip;
    return( l_image );
}

static ARInt16 *labeling3( ARUint8 *image, int thresh,
                    int *label_num, int **area, double **pos, int **clip,
                    int **label_ref, int LorR )
{
    ARUint8   *pnt;                 /*  image pointer       */
    ARInt16   *pnt1, *pnt2;         /*  image pointer       */
    int       *wk;                  /*  pointer for work    */
    int       wk_max;               /*  work                */
    int       m,n;                  /*  work                */
    int       i,j,k;                /*  for loop            */
    int       lxsize, lysize;
    int       poff;
    ARUint8   *dpnt;
    ARInt16   *l_image;
    int       *work, *work2;
```

```
    int      *wlabel_num;
    int      *warea;
    int      *wclip;
    double   *wpos;

    thresh *= 3;
    if( arImageProcMode == AR_IMAGE_PROC_IN_HALF ) {
        lxsize = arImXsize / 2;
        lysize = arImYsize / 2;
    }
    else {
        lxsize = arImXsize;
        lysize = arImYsize;
    }

    if( LorR ) {
        l_image = &l_imageL[0];
        work    = &workL[0];
        work2   = &work2L[0];
        wlabel_num = &wlabel_numL;
        warea   = &wareaL[0];
        wclip   = &wclipL[0];
        wpos    = &wposL[0];
        if( arImageL == NULL ) {
#if 0
            int texXsize = 1;
            int texYsize = 1;
            while( texXsize < arImXsize ) texXsize *= 2;
            if( texXsize > 512 ) texXsize = 512;
            while( texYsize < arImYsize ) texYsize *= 2;
            arMalloc( arImageL, ARUint8, texXsize*texYsize*AR_PIX_SIZE );
#else
            arMalloc( arImageL, ARUint8, arImXsize*arImYsize*AR_PIX_SIZE );
#endif
            put_zero( arImageL, lxsize*lysize*AR_PIX_SIZE );
            arImage = arImageL;
        }
    }
    else {
        l_image = &l_imageR[0];
        work    = &workR[0];
        work2   = &work2R[0];
        wlabel_num = &wlabel_numR;
        warea   = &wareaR[0];
        wclip   = &wclipR[0];
        wpos    = &wposR[0];
        if( arImageR == NULL ) {
#if 0
            int texXsize = 1;
            int texYsize = 1;
            while( texXsize < arImXsize ) texXsize *= 2;
            if( texXsize > 512 ) texXsize = 512;
            while( texYsize < arImYsize ) texYsize *= 2;
```

```
            arMalloc( arImageR, ARUint8, texXsize*texYsize*AR_PIX_SIZE );
#else
            arMalloc( arImageR, ARUint8, arImXsize*arImYsize*AR_PIX_SIZE );
#endif
            put_zero( arImageR, lxsize*lysize*AR_PIX_SIZE );
        }
    }

    pnt1 = &l_image[0];
    pnt2 = &l_image[(lysize-1)*lxsize];
    for(i = 0; i < lxsize; i++) {
        *(pnt1++) = *(pnt2++) = 0;
    }

    pnt1 = &l_image[0];
    pnt2 = &l_image[lxsize-1];
    for(i = 0; i < lysize; i++) {
        *pnt1 = *pnt2 = 0;
        pnt1 += lxsize;
        pnt2 += lxsize;
    }

    wk_max = 0;
    pnt2 = &(l_image[lxsize+1]);
    if( LorR ) dpnt = &(arImageL[(lxsize+1)*AR_PIX_SIZE]);
    else       dpnt = &(arImageR[(lxsize+1)*AR_PIX_SIZE]);
    if( arImageProcMode == AR_IMAGE_PROC_IN_HALF ) {
        pnt = &(image[(arImXsize*2+2)*AR_PIX_SIZE]);
        poff = AR_PIX_SIZE*2;
    }
    else {
        pnt = &(image[(arImXsize+1)*AR_PIX_SIZE]);
        poff = AR_PIX_SIZE;
    }
    for(j = 1; j < lysize-1; j++, pnt+=poff*2, pnt2+=2, dpnt+=AR_PIX_SIZE*2) {
        for(i = 1; i < lxsize-1; i++, pnt+=poff, pnt2++, dpnt+=AR_PIX_SIZE) {
#if defined(AR_PIX_FORMAT_ARGB)
            if( *(pnt+1) + *(pnt+2) + *(pnt+3) <= thresh ) {
                *(dpnt+1) = *(dpnt+2) = *(dpnt+3) = 255;
#elif defined(AR_PIX_FORMAT_ABGR)
            if( *(pnt+1) + *(pnt+2) + *(pnt+3) <= thresh ) {
                *(dpnt+1) = *(dpnt+2) = *(dpnt+3) = 255;
#elif defined(AR_PIX_FORMAT_BGRA)
            if( *(pnt+0) + *(pnt+1) + *(pnt+2) <= thresh ) {
                *(dpnt+0) = *(dpnt+1) = *(dpnt+2) = 255;
#elif defined(AR_PIX_FORMAT_BGR)
            if( *(pnt+0) + *(pnt+1) + *(pnt+2) <= thresh ) {
                *(dpnt+0) = *(dpnt+1) = *(dpnt+2) = 255;
#elif defined(AR_PIX_FORMAT_RGBA)
            if( *(pnt+0) + *(pnt+1) + *(pnt+2) <= thresh ) {
                *(dpnt+0) = *(dpnt+1) = *(dpnt+2) = 255;
#elif defined(AR_PIX_FORMAT_RGB)
            if( *(pnt+0) + *(pnt+1) + *(pnt+2) <= thresh ) {
```

```
            *(dpnt+0) = *(dpnt+1) = *(dpnt+2) = 255;
#elif defined(AR_PIX_FORMAT_2vuy)
                        if( *(pnt+1) * 3 <= thresh ) {
                                *(dpnt+1) = 255;
#elif defined(AR_PIX_FORMAT_yuvs)
                        if( *(pnt+0) * 3 <= thresh ) {
                                *(dpnt+0) = 255;
#else
#  error Unknown pixel format defined in config.h
#endif
                                                        pnt1 = &(pnt2[-lxsize]);
            if( *pnt1 > 0 ) {
               *pnt2 = *pnt1;
               work2[((*pnt2)-1)*7+0] ++;
               work2[((*pnt2)-1)*7+1] += i;
               work2[((*pnt2)-1)*7+2] += j;
               work2[((*pnt2)-1)*7+6] = j;
            }
            else if( *(pnt1+1) > 0 ) {
               if( *(pnt1-1) > 0 ) {
                  m = work[*(pnt1+1)-1];
                  n = work[*(pnt1-1)-1];
                  if( m > n ) {
                     *pnt2 = n;
                     wk = &(work[0]);
                     for(k = 0; k < wk_max; k++) {
                        if( *wk == m ) *wk = n;
                        wk++;
                     }
                  }
                  else if( m < n ) {
                     *pnt2 = m;
                     wk = &(work[0]);
                     for(k = 0; k < wk_max; k++) {
                        if( *wk == n ) *wk = m;
                        wk++;
                     }
                  }
                  else *pnt2 = m;
                  work2[((*pnt2)-1)*7+0] ++;
                  work2[((*pnt2)-1)*7+1] += i;
                  work2[((*pnt2)-1)*7+2] += j;
                  work2[((*pnt2)-1)*7+6] = j;
               }
               else if( *(pnt2-1) > 0 ) {
                  m = work[*(pnt1+1)-1];
                  n = work[*(pnt2-1)-1];
                  if( m > n ) {
                     *pnt2 = n;
                     wk = &(work[0]);
                     for(k = 0; k < wk_max; k++) {
                        if( *wk == m ) *wk = n;
                        wk++;
```

```
            }
        }
        else if( m < n ) {
            *pnt2 = m;
            wk = &(work[0]);
            for(k = 0; k < wk_max; k++) {
                if( *wk == n ) *wk = m;
                wk++;
            }
        }
        else *pnt2 = m;
        work2[((*pnt2)-1)*7+0] ++;
        work2[((*pnt2)-1)*7+1] += i;
        work2[((*pnt2)-1)*7+2] += j;
    }
    else {
        *pnt2 = *(pnt1+1);
        work2[((*pnt2)-1)*7+0] ++;
        work2[((*pnt2)-1)*7+1] += i;
        work2[((*pnt2)-1)*7+2] += j;
        if( work2[((*pnt2)-1)*7+3] > i ) work2[((*pnt2)-1)*7+3] = i;
        work2[((*pnt2)-1)*7+6] = j;
    }
}
else if( *(pnt1-1) > 0 ) {
    *pnt2 = *(pnt1-1);
    work2[((*pnt2)-1)*7+0] ++;
    work2[((*pnt2)-1)*7+1] += i;
    work2[((*pnt2)-1)*7+2] += j;
    if( work2[((*pnt2)-1)*7+4] < i ) work2[((*pnt2)-1)*7+4] = i;
    work2[((*pnt2)-1)*7+6] = j;
}
else if( *(pnt2-1) > 0) {
    *pnt2 = *(pnt2-1);
    work2[((*pnt2)-1)*7+0] ++;
    work2[((*pnt2)-1)*7+1] += i;
    work2[((*pnt2)-1)*7+2] += j;
    if( work2[((*pnt2)-1)*7+4] < i ) work2[((*pnt2)-1)*7+4] = i;
}
else {
    wk_max++;
    if( wk_max > WORK_SIZE ) {
        return(0);
    }
    work[wk_max-1] = *pnt2 = wk_max;
    work2[(wk_max-1)*7+0] = 1;
    work2[(wk_max-1)*7+1] = i;
    work2[(wk_max-1)*7+2] = j;
    work2[(wk_max-1)*7+3] = i;
    work2[(wk_max-1)*7+4] = i;
    work2[(wk_max-1)*7+5] = j;
    work2[(wk_max-1)*7+6] = j;
}
```

```
            }
            else {
                *pnt2 = 0;
#if defined(AR_PIX_FORMAT_ARGB)
                *(dpnt+1) = *(dpnt+2) = *(dpnt+3) = 0;
#elif defined(AR_PIX_FORMAT_ABGR)
                *(dpnt+1) = *(dpnt+2) = *(dpnt+3) = 0;
#elif defined(AR_PIX_FORMAT_BGRA)
                *(dpnt+0) = *(dpnt+1) = *(dpnt+2) = 0;
#elif defined(AR_PIX_FORMAT_BGR)
                *(dpnt+0) = *(dpnt+1) = *(dpnt+2) = 0;
#elif defined(AR_PIX_FORMAT_RGBA)
                *(dpnt+0) = *(dpnt+1) = *(dpnt+2) = 0;
#elif defined(AR_PIX_FORMAT_RGB)
                *(dpnt+0) = *(dpnt+1) = *(dpnt+2) = 0;
#elif defined(AR_PIX_FORMAT_2vuy)
                *(dpnt+1) = 0;
#elif defined(AR_PIX_FORMAT_yuvs)
                *(dpnt+0) = 0;
#else
#  error Unknown pixel format defined in config.h
#endif
            }
        }
        if( arImageProcMode == AR_IMAGE_PROC_IN_HALF ) pnt +=
arImXsize*AR_PIX_SIZE;
    }

    j = 1;
    wk = &(work[0]);
    for(i = 1; i <= wk_max; i++, wk++) {
        *wk = (*wk==i)? j++: work[(*wk)-1];
    }
    *label_num = *wlabel_num = j - 1;
    if( *label_num == 0 ) {
        return( l_image );
    }

    put_zero( (ARUint8 *)warea, *label_num *     sizeof(int) );
    put_zero( (ARUint8 *)wpos,  *label_num * 2 * sizeof(double) );
    for(i = 0; i < *label_num; i++) {
        wclip[i*4+0] = lxsize;
        wclip[i*4+1] = 0;
        wclip[i*4+2] = lysize;
        wclip[i*4+3] = 0;
    }
    for(i = 0; i < wk_max; i++) {
        j = work[i] - 1;
        warea[j]    += work2[i*7+0];
        wpos[j*2+0] += work2[i*7+1];
        wpos[j*2+1] += work2[i*7+2];
        if( wclip[j*4+0] > work2[i*7+3] ) wclip[j*4+0] = work2[i*7+3];
        if( wclip[j*4+1] < work2[i*7+4] ) wclip[j*4+1] = work2[i*7+4];
```

```
        if( wclip[j*4+2] > work2[i*7+5] ) wclip[j*4+2] = work2[i*7+5];
        if( wclip[j*4+3] < work2[i*7+6] ) wclip[j*4+3] = work2[i*7+6];
    }

    for( i = 0; i < *label_num; i++ ) {
        wpos[i*2+0] /= warea[i];
        wpos[i*2+1] /= warea[i];
    }

    *label_ref = work;
    *area      = warea;
    *pos       = wpos;
    *clip      = wclip;
    return( l_image );
}



ARInt16 *arLabelingHSB( ARUint8 *image, int thresh,
                        int *label_num, int **area, double **pos, int **clip,
                        int **label_ref, int LorR, int minH, int maxH, int minS, int maxS, int minB, int
maxB, int invert)
{
    ARUint8   *pnt;                 /*  image pointer       */
    ARInt16   *pnt1, *pnt2;         /*  image pointer       */
    int       *wk;                  /*  pointer for work    */
    int       wk_max;               /*  work                */
    int       m,n;                  /*  work                */
    int       i,j,k;                /*  for loop            */
    int       lxsize, lysize;
    int       poff;
    ARInt16   *l_image;
    int       *work, *work2;
    int       *wlabel_num;
    int       *warea;
    int       *wclip;
    double    *wpos;
          double  H, S, V;

#ifdef USE_OPTIMIZATIONS
          int                   pnt2_index;   // [tp]
#endif

          if( LorR ) {
        l_image = &l_imageL[0];
        work    = &workL[0];
        work2   = &work2L[0];
        wlabel_num = &wlabel_numL;
        warea   = &wareaL[0];
        wclip   = &wclipL[0];
        wpos    = &wposL[0];
    }
    else {
```

```
        l_image = &l_imageR[0];
        work    = &workR[0];
        work2   = &work2R[0];
        wlabel_num = &wlabel_numR;
        warea   = &wareaR[0];
        wclip   = &wclipR[0];
        wpos    = &wposR[0];
    }

    thresh *= 3;
    if( arImageProcMode == AR_IMAGE_PROC_IN_HALF ) {
        lxsize = arImXsize / 2;
        lysize = arImYsize / 2;
    }
    else {
        lxsize = arImXsize;
        lysize = arImYsize;
    }

    pnt1 = &l_image[0];
    pnt2 = &l_image[(lysize-1)*lxsize];

#ifndef USE_OPTIMIZATIONS
            for(i = 0; i < lxsize; i++) {
        *(pnt1++) = *(pnt2++) = 0;
    }
#else
// 4x loop unrolling
            for(i = 0; i < lxsize-(lxsize%4); i+=4) {
        *(pnt1++) = *(pnt2++) = 0;
        *(pnt1++) = *(pnt2++) = 0;
        *(pnt1++) = *(pnt2++) = 0;
        *(pnt1++) = *(pnt2++) = 0;
    }
#endif
    pnt1 = &l_image[0];
    pnt2 = &l_image[lxsize-1];

#ifndef USE_OPTIMIZATIONS
    for(i = 0; i < lysize; i++) {
        *pnt1 = *pnt2 = 0;
        pnt1 += lxsize;
        pnt2 += lxsize;
    }
#else
// 4x loop unrolling
    for(i = 0; i < lysize-(lysize%4); i+=4) {
                    *pnt1 = *pnt2 = 0;
        pnt1 += lxsize;
        pnt2 += lxsize;

                    *pnt1 = *pnt2 = 0;
        pnt1 += lxsize;
```

```
        pnt2 += lxsize;

                *pnt1 = *pnt2 = 0;
        pnt1 += lxsize;
        pnt2 += lxsize;

                *pnt1 = *pnt2 = 0;
        pnt1 += lxsize;
        pnt2 += lxsize;
    }
#endif

    wk_max = 0;
    pnt2 = &(l_image[lxsize+1]);
    if( arImageProcMode == AR_IMAGE_PROC_IN_HALF ) {
        pnt = &(image[(arImXsize*2+2)*AR_PIX_SIZE]);
        poff = AR_PIX_SIZE*2;
    }
    else {
        pnt = &(image[(arImXsize+1)*AR_PIX_SIZE]);
        poff = AR_PIX_SIZE;
    }
    for(j = 1; j < lysize-1; j++, pnt+=poff*2, pnt2+=2) {
        for(i = 1; i < lxsize-1; i++, pnt+=poff, pnt2++) {


  RGBtoHSV(R_COMP,G_COMP,B_COMP,&H,&S,&V);
  if((((H>=minH && H<=maxH) && (S>=minS && S<=maxS) && (V>=minB && V<=maxB))
&& !invert) || (!((H>=minH && H<=maxH) && (S>=minS && S<=maxS) && (V>=minB &&
V<=maxB)) && invert))  {


            pnt1 = &(pnt2[-lxsize]);
            if( *pnt1 > 0 ) {
                *pnt2 = *pnt1;

#ifndef USE_OPTIMIZATIONS
                                        // ORIGINAL CODE
                                        work2[((*pnt2)-1)*7+0] ++;
                work2[((*pnt2)-1)*7+1] += i;
                work2[((*pnt2)-1)*7+2] += j;
                work2[((*pnt2)-1)*7+6] = j;
#else
                                        // OPTIMIZED CODE [tp]
                                        // ((*pnt2)-1)*7 should be treated as constant, since
                                        //  work2[n] (n=0..xsize*ysize) cannot overwrite (*pnt2)
                                        pnt2_index = ((*pnt2)-1) * 7;
                work2[pnt2_index+0]++;
                work2[pnt2_index+1]+= i;
                work2[pnt2_index+2]+= j;
                work2[pnt2_index+6] = j;
                                        // --------------------------------
#endif
```

```
            }
            else if( *(pnt1+1) > 0 ) {
               if( *(pnt1-1) > 0 ) {
                  m = work[*(pnt1+1)-1];
                  n = work[*(pnt1-1)-1];
                  if( m > n ) {
                     *pnt2 = n;
                     wk = &(work[0]);
                     for(k = 0; k < wk_max; k++) {
                        if( *wk == m ) *wk = n;
                        wk++;
                     }
                  }
                  else if( m < n ) {
                     *pnt2 = m;
                     wk = &(work[0]);
                     for(k = 0; k < wk_max; k++) {
                        if( *wk == n ) *wk = m;
                        wk++;
                     }
                  }
                  else *pnt2 = m;

#ifndef USE_OPTIMIZATIONS
                                                   // ORIGINAL CODE
                                                   work2[((*pnt2)-1)*7+0] ++;
                  work2[((*pnt2)-1)*7+1] += i;
                  work2[((*pnt2)-1)*7+2] += j;
                  work2[((*pnt2)-1)*7+6] = j;
#else
                                                   // PERFORMANCE OPTIMIZATION:
                                                   pnt2_index = ((*pnt2)-1) * 7;
                                                   work2[pnt2_index+0]++;
                                                   work2[pnt2_index+1]+= i;
                                                   work2[pnt2_index+2]+= j;
                                                   work2[pnt2_index+6] = j;
#endif

               }
               else if( *(pnt2-1) > 0 ) {
                  m = work[*(pnt1+1)-1];
                  n = work[*(pnt2-1)-1];
                  if( m > n ) {
                     *pnt2 = n;
                     wk = &(work[0]);
                     for(k = 0; k < wk_max; k++) {
                        if( *wk == m ) *wk = n;
                        wk++;
                     }
                  }
                  else if( m < n ) {
                     *pnt2 = m;
                     wk = &(work[0]);
```

```
                    for(k = 0; k < wk_max; k++) {
                        if( *wk == n ) *wk = m;
                        wk++;
                    }
                }
                else *pnt2 = m;

#ifndef USE_OPTIMIZATIONS
                                                // ORIGINAL CODE
                work2[((*pnt2)-1)*7+0] ++;
                work2[((*pnt2)-1)*7+1] += i;
                work2[((*pnt2)-1)*7+2] += j;
#else
                                                // PERFORMANCE OPTIMIZATION:
                                                pnt2_index = ((*pnt2)-1) * 7;
                                                work2[pnt2_index+0]++;
                                                work2[pnt2_index+1]+= i;
                                                work2[pnt2_index+2]+= j;
#endif

            }
            else {
                *pnt2 = *(pnt1+1);

#ifndef USE_OPTIMIZATIONS
                                                // ORIGINAL CODE
                work2[((*pnt2)-1)*7+0] ++;
                work2[((*pnt2)-1)*7+1] += i;
                work2[((*pnt2)-1)*7+2] += j;
                if( work2[((*pnt2)-1)*7+3] > i ) work2[((*pnt2)-1)*7+3] = i;
                work2[((*pnt2)-1)*7+6] = j;
#else
                                                // PERFORMANCE OPTIMIZATION:
                                                pnt2_index = ((*pnt2)-1) * 7;
                                                work2[pnt2_index+0]++;
                                                work2[pnt2_index+1]+= i;
                                                work2[pnt2_index+2]+= j;
                if( work2[pnt2_index+3] > i ) work2[pnt2_index+3] = i;
                                                work2[pnt2_index+6] = j;
#endif
            }
        }
        else if( *(pnt1-1) > 0 ) {
            *pnt2 = *(pnt1-1);

#ifndef USE_OPTIMIZATIONS
                                                // ORIGINAL CODE
            work2[((*pnt2)-1)*7+0] ++;
            work2[((*pnt2)-1)*7+1] += i;
            work2[((*pnt2)-1)*7+2] += j;
            if( work2[((*pnt2)-1)*7+4] < i ) work2[((*pnt2)-1)*7+4] = i;
            work2[((*pnt2)-1)*7+6] = j;
#else
```

```
                                        // PERFORMANCE OPTIMIZATION:
                                        pnt2_index = ((*pnt2)-1) * 7;
                                        work2[pnt2_index+0]++;
                                        work2[pnt2_index+1]+= i;
                                        work2[pnt2_index+2]+= j;
                if( work2[pnt2_index+4] < i ) work2[pnt2_index+4] = i;
                                        work2[pnt2_index+6] = j;
#endif
            }
            else if( *(pnt2-1) > 0) {
                *pnt2 = *(pnt2-1);

#ifndef USE_OPTIMIZATIONS
                                                // ORIGINAL CODE
                work2[((*pnt2)-1)*7+0] ++;
                work2[((*pnt2)-1)*7+1] += i;
                work2[((*pnt2)-1)*7+2] += j;
                if( work2[((*pnt2)-1)*7+4] < i ) work2[((*pnt2)-1)*7+4] = i;
#else
                                        // PERFORMANCE OPTIMIZATION:
                                        pnt2_index = ((*pnt2)-1) * 7;
                                        work2[pnt2_index+0]++;
                                        work2[pnt2_index+1]+= i;
                                        work2[pnt2_index+2]+= j;
                if( work2[pnt2_index+4] < i ) work2[pnt2_index+4] = i;
#endif
                                    }
            else {
                wk_max++;
                if( wk_max > WORK_SIZE ) {
                    return(0);
                }
                work[wk_max-1] = *pnt2 = wk_max;
                work2[(wk_max-1)*7+0] = 1;
                work2[(wk_max-1)*7+1] = i;
                work2[(wk_max-1)*7+2] = j;
                work2[(wk_max-1)*7+3] = i;
                work2[(wk_max-1)*7+4] = i;
                work2[(wk_max-1)*7+5] = j;
                work2[(wk_max-1)*7+6] = j;
            }
        }
        else {
            *pnt2 = 0;
        }
    }
    if( arImageProcMode == AR_IMAGE_PROC_IN_HALF ) pnt +=
arImXsize*AR_PIX_SIZE;
  }

  j = 1;
  wk = &(work[0]);
  for(i = 1; i <= wk_max; i++, wk++) {
```

```
        *wk = (*wk==i)? j++: work[(*wk)-1];
    }
    *label_num = *wlabel_num = j - 1;
    if( *label_num == 0 ) {
        return( l_image );
    }

    put_zero( (ARUint8 *)warea, *label_num *     sizeof(int) );
    put_zero( (ARUint8 *)wpos,  *label_num * 2 * sizeof(double) );
    for(i = 0; i < *label_num; i++) {
        wclip[i*4+0] = lxsize;
        wclip[i*4+1] = 0;
        wclip[i*4+2] = lysize;
        wclip[i*4+3] = 0;
    }
    for(i = 0; i < wk_max; i++) {
        j = work[i] - 1;
        warea[j]    += work2[i*7+0];
        wpos[j*2+0] += work2[i*7+1];
        wpos[j*2+1] += work2[i*7+2];
        if( wclip[j*4+0] > work2[i*7+3] ) wclip[j*4+0] = work2[i*7+3];
        if( wclip[j*4+1] < work2[i*7+4] ) wclip[j*4+1] = work2[i*7+4];
        if( wclip[j*4+2] > work2[i*7+5] ) wclip[j*4+2] = work2[i*7+5];
        if( wclip[j*4+3] < work2[i*7+6] ) wclip[j*4+3] = work2[i*7+6];
    }

    for( i = 0; i < *label_num; i++ ) {
        wpos[i*2+0] /= warea[i];
        wpos[i*2+1] /= warea[i];
    }

    *label_ref = work;
    *area      = warea;
    *pos       = wpos;
    *clip      = wclip;
    return( l_image );
}




ARInt16 *arLabelingHSBMultiple( ARUint8 *image, int thresh,
                        int *label_num, int **area, double **pos, int **clip,
                        int **label_ref, int LorR, int *minH, int *maxH, int *minS, int *maxS, int *minB,
int *maxB, int numHSV, int invert)
{
    ARUint8   *pnt;                 /*  image pointer       */
    ARInt16   *pnt1, *pnt2;         /*  image pointer       */
    int       *wk;                  /*  pointer for work    */
    int       wk_max;               /*  work                */
    int       m,n;                  /*  work                */
    int       i,j,k, l;             /*  for loop            */
    int       lxsize, lysize;
```

```
    int      poff;
    ARInt16  *l_image;
    int      *work, *work2;
    int      *wlabel_num;
    int      *warea;
    int      *wclip;
    double   *wpos;
          double  H, S, V;
          int detectado;

#ifdef USE_OPTIMIZATIONS
          int                 pnt2_index;   // [tp]
#endif

          if( LorR ) {
       l_image = &l_imageL[0];
       work    = &workL[0];
       work2   = &work2L[0];
       wlabel_num = &wlabel_numL;
       warea   = &wareaL[0];
       wclip   = &wclipL[0];
       wpos    = &wposL[0];
    }
    else {
       l_image = &l_imageR[0];
       work    = &workR[0];
       work2   = &work2R[0];
       wlabel_num = &wlabel_numR;
       warea   = &wareaR[0];
       wclip   = &wclipR[0];
       wpos    = &wposR[0];
    }

    thresh *= 3;
    if( arImageProcMode == AR_IMAGE_PROC_IN_HALF ) {
       lxsize = arImXsize / 2;
       lysize = arImYsize / 2;
    }
    else {
       lxsize = arImXsize;
       lysize = arImYsize;
    }

    pnt1 = &l_image[0];
    pnt2 = &l_image[(lysize-1)*lxsize];

#ifndef USE_OPTIMIZATIONS
          for(i = 0; i < lxsize; i++) {
       *(pnt1++) = *(pnt2++) = 0;
    }
#else
// 4x loop unrolling
          for(i = 0; i < lxsize-(lxsize%4); i+=4) {
```

```
        *(pnt1++) = *(pnt2++) = 0;
        *(pnt1++) = *(pnt2++) = 0;
        *(pnt1++) = *(pnt2++) = 0;
        *(pnt1++) = *(pnt2++) = 0;
    }
#endif
    pnt1 = &l_image[0];
    pnt2 = &l_image[lxsize-1];

#ifndef USE_OPTIMIZATIONS
    for(i = 0; i < lysize; i++) {
        *pnt1 = *pnt2 = 0;
        pnt1 += lxsize;
        pnt2 += lxsize;
    }
#else
// 4x loop unrolling
    for(i = 0; i < lysize-(lysize%4); i+=4) {
                    *pnt1 = *pnt2 = 0;
        pnt1 += lxsize;
        pnt2 += lxsize;

                    *pnt1 = *pnt2 = 0;
        pnt1 += lxsize;
        pnt2 += lxsize;

                    *pnt1 = *pnt2 = 0;
        pnt1 += lxsize;
        pnt2 += lxsize;

                    *pnt1 = *pnt2 = 0;
        pnt1 += lxsize;
        pnt2 += lxsize;
    }
#endif

    wk_max = 0;
    pnt2 = &(l_image[lxsize+1]);
    if( arImageProcMode == AR_IMAGE_PROC_IN_HALF ) {
        pnt = &(image[(arImXsize*2+2)*AR_PIX_SIZE]);
        poff = AR_PIX_SIZE*2;
    }
    else {
        pnt = &(image[(arImXsize+1)*AR_PIX_SIZE]);
        poff = AR_PIX_SIZE;
    }
    for(j = 1; j < lysize-1; j++, pnt+=poff*2, pnt2+=2) {
        for(i = 1; i < lxsize-1; i++, pnt+=poff, pnt2++) {

    detectado=0;
          RGBtoHSV(R_COMP,G_COMP,B_COMP,&H,&S,&V);
          for(l=0;l<numHSV;l++) {
                  if((H>=*(minH+l) && H<=*(maxH+l)) && (S>=*(minS+l) && S<=*(maxS+l))
```

```
&& (V>=*(minB+l) && V<=*(maxB+l)))  {
                    detectado=1;
                    break;
                   }
          }



  if(detectado)  {
      // printf("H=%f S=%f V=%f, ",H,S,V);

        //  if( *(pnt+0) + *(pnt+1) + *(pnt+2) <= thresh ) { //original

            pnt1 = &(pnt2[-lxsize]);
            if( *pnt1 > 0 ) {
               *pnt2 = *pnt1;

#ifndef USE_OPTIMIZATIONS
                                        // ORIGINAL CODE
                                        work2[((*pnt2)-1)*7+0] ++;
               work2[((*pnt2)-1)*7+1] += i;
               work2[((*pnt2)-1)*7+2] += j;
               work2[((*pnt2)-1)*7+6] = j;
#else
                                        // OPTIMIZED CODE [tp]
                                        // ((*pnt2)-1)*7 should be treated as constant, since
                                        //  work2[n] (n=0..xsize*ysize) cannot overwrite (*pnt2)
                                        pnt2_index = ((*pnt2)-1) * 7;
               work2[pnt2_index+0]++;
               work2[pnt2_index+1]+= i;
               work2[pnt2_index+2]+= j;
               work2[pnt2_index+6] = j;
                                        // --------------------------------
#endif
            }
            else if( *(pnt1+1) > 0 ) {
               if( *(pnt1-1) > 0 ) {
                  m = work[*(pnt1+1)-1];
                  n = work[*(pnt1-1)-1];
                  if( m > n ) {
                     *pnt2 = n;
                     wk = &(work[0]);
                     for(k = 0; k < wk_max; k++) {
                        if( *wk == m ) *wk = n;
                        wk++;
                     }
                  }
                  else if( m < n ) {
                     *pnt2 = m;
                     wk = &(work[0]);
                     for(k = 0; k < wk_max; k++) {
                        if( *wk == n ) *wk = m;
                        wk++;
```

```
                }
            }
            else *pnt2 = m;

#ifndef USE_OPTIMIZATIONS
                                        // ORIGINAL CODE
                                        work2[((*pnt2)-1)*7+0] ++;
            work2[((*pnt2)-1)*7+1] += i;
            work2[((*pnt2)-1)*7+2] += j;
            work2[((*pnt2)-1)*7+6] = j;
#else
                                        // PERFORMANCE OPTIMIZATION:
                                        pnt2_index = ((*pnt2)-1) * 7;
                                        work2[pnt2_index+0]++;
                                        work2[pnt2_index+1]+= i;
                                        work2[pnt2_index+2]+= j;
                                        work2[pnt2_index+6] = j;
#endif

        }
        else if( *(pnt2-1) > 0 ) {
            m = work[*(pnt1+1)-1];
            n = work[*(pnt2-1)-1];
            if( m > n ) {
                *pnt2 = n;
                wk = &(work[0]);
                for(k = 0; k < wk_max; k++) {
                    if( *wk == m ) *wk = n;
                    wk++;
                }
            }
            else if( m < n ) {
                *pnt2 = m;
                wk = &(work[0]);
                for(k = 0; k < wk_max; k++) {
                    if( *wk == n ) *wk = m;
                    wk++;
                }
            }
            else *pnt2 = m;

#ifndef USE_OPTIMIZATIONS
                                        // ORIGINAL CODE
            work2[((*pnt2)-1)*7+0] ++;
            work2[((*pnt2)-1)*7+1] += i;
            work2[((*pnt2)-1)*7+2] += j;
#else
                                        // PERFORMANCE OPTIMIZATION:
                                        pnt2_index = ((*pnt2)-1) * 7;
                                        work2[pnt2_index+0]++;
                                        work2[pnt2_index+1]+= i;
                                        work2[pnt2_index+2]+= j;
#endif
```

```
            }
            else {
               *pnt2 = *(pnt1+1);

#ifndef USE_OPTIMIZATIONS
                                                // ORIGINAL CODE
               work2[((*pnt2)-1)*7+0] ++;
               work2[((*pnt2)-1)*7+1] += i;
               work2[((*pnt2)-1)*7+2] += j;
               if( work2[((*pnt2)-1)*7+3] > i ) work2[((*pnt2)-1)*7+3] = i;
               work2[((*pnt2)-1)*7+6] = j;
#else
                                                // PERFORMANCE OPTIMIZATION:
                                                pnt2_index = ((*pnt2)-1) * 7;
                                                work2[pnt2_index+0]++;
                                                work2[pnt2_index+1]+= i;
                                                work2[pnt2_index+2]+= j;
               if( work2[pnt2_index+3] > i ) work2[pnt2_index+3] = i;
                                                work2[pnt2_index+6] = j;
#endif
            }
         }
         else if( *(pnt1-1) > 0 ) {
            *pnt2 = *(pnt1-1);

#ifndef USE_OPTIMIZATIONS
                                                // ORIGINAL CODE
            work2[((*pnt2)-1)*7+0] ++;
            work2[((*pnt2)-1)*7+1] += i;
            work2[((*pnt2)-1)*7+2] += j;
            if( work2[((*pnt2)-1)*7+4] < i ) work2[((*pnt2)-1)*7+4] = i;
            work2[((*pnt2)-1)*7+6] = j;
#else
                                        // PERFORMANCE OPTIMIZATION:
                                        pnt2_index = ((*pnt2)-1) * 7;
                                        work2[pnt2_index+0]++;
                                        work2[pnt2_index+1]+= i;
                                        work2[pnt2_index+2]+= j;
            if( work2[pnt2_index+4] < i ) work2[pnt2_index+4] = i;
                                        work2[pnt2_index+6] = j;
#endif
         }
         else if( *(pnt2-1) > 0) {
            *pnt2 = *(pnt2-1);

#ifndef USE_OPTIMIZATIONS
                                                // ORIGINAL CODE
            work2[((*pnt2)-1)*7+0] ++;
            work2[((*pnt2)-1)*7+1] += i;
            work2[((*pnt2)-1)*7+2] += j;
            if( work2[((*pnt2)-1)*7+4] < i ) work2[((*pnt2)-1)*7+4] = i;
#else
```

```
                                        // PERFORMANCE OPTIMIZATION:
                                        pnt2_index = ((*pnt2)-1) * 7;
                                        work2[pnt2_index+0]++;
                                        work2[pnt2_index+1]+= i;
                                        work2[pnt2_index+2]+= j;
                if( work2[pnt2_index+4] < i ) work2[pnt2_index+4] = i;
#endif
                                    }
            else {
               wk_max++;
               if( wk_max > WORK_SIZE ) {
                  return(0);
               }
               work[wk_max-1] = *pnt2 = wk_max;
               work2[(wk_max-1)*7+0] = 1;
               work2[(wk_max-1)*7+1] = i;
               work2[(wk_max-1)*7+2] = j;
               work2[(wk_max-1)*7+3] = i;
               work2[(wk_max-1)*7+4] = i;
               work2[(wk_max-1)*7+5] = j;
               work2[(wk_max-1)*7+6] = j;
            }
         }
         else {
            *pnt2 = 0;
         }
      }
      if( arImageProcMode == AR_IMAGE_PROC_IN_HALF ) pnt +=
arImXsize*AR_PIX_SIZE;
   }

   j = 1;
   wk = &(work[0]);
   for(i = 1; i <= wk_max; i++, wk++) {
      *wk = (*wk==i)? j++: work[(*wk)-1];
   }
   *label_num = *wlabel_num = j - 1;
   if( *label_num == 0 ) {
      return( l_image );
   }

   put_zero( (ARUint8 *)warea, *label_num *     sizeof(int) );
   put_zero( (ARUint8 *)wpos,  *label_num * 2 * sizeof(double) );
   for(i = 0; i < *label_num; i++) {
      wclip[i*4+0] = lxsize;
      wclip[i*4+1] = 0;
      wclip[i*4+2] = lysize;
      wclip[i*4+3] = 0;
   }
   for(i = 0; i < wk_max; i++) {
      j = work[i] - 1;
      warea[j]    += work2[i*7+0];
      wpos[j*2+0] += work2[i*7+1];
```

```
        wpos[j*2+1] += work2[i*7+2];
        if( wclip[j*4+0] > work2[i*7+3] ) wclip[j*4+0] = work2[i*7+3];
        if( wclip[j*4+1] < work2[i*7+4] ) wclip[j*4+1] = work2[i*7+4];
        if( wclip[j*4+2] > work2[i*7+5] ) wclip[j*4+2] = work2[i*7+5];
        if( wclip[j*4+3] < work2[i*7+6] ) wclip[j*4+3] = work2[i*7+6];
    }

    for( i = 0; i < *label_num; i++ ) {
        wpos[i*2+0] /= warea[i];
        wpos[i*2+1] /= warea[i];
    }

    *label_ref = work;
    *area      = warea;
    *pos       = wpos;
    *clip      = wclip;
    return( l_image );
}
```

Fichero arColor.c

```
#include <stdlib.h>
#include <string.h>
#include <math.h>
#ifndef _WIN32
#define max(a,b) ((a)>(b) ? (a) : (b))
#define min(a,b) ((a)<(b) ? (a) : (b))
#endif


//funcion para transformar RBB a HSV
void RGBtoHSV(double r, double g, double b, double *H, double *S, double *V) {
  double D;
  double min;
  min = min(min(r, g),b);
  *V = max(max(r, g),b);

  D = *V - min;

  if(*V == 0.0) *S = 0;
   else *S = D / *V;

  if(*S == 0.0)  *H = 0;
    else if (r == *V) *H = 60.0 * (g - b) / D;
     else if(g == *V)  *H = 120.0 + 60.0 * (b - r) / D;
       else if (b == *V) *H = 240.0 + 60.0 * (r - g) / D;
  if (*H < 0.0) *H = *H + 360.0;

  *S=*(S) * 100; //conversion de escala 0 a 1, a escala 0 a 100
  *V=*(V)/255 * 100;//conversion de escala 0 a 255, a escala 0 a 100

}
```

Fichero ar.h

```
------------------------------------------
*   Copyright (C) 2004 Hitlab NZ.
*   The distribution policy is describe on the Copyright.txt furnish
*    with this library.
*   ----------------------------------------------------------------------*/
/**
*  \file ar.h
*  \brief ARToolKit subroutines.
*
*  Core of the ARToolKit Library. This file provides image analysis and marker
*  detection routines. Differents routines give access to camera and marker
*  configurations. Other routines manipulate marker info structures for
*  deliver 3D transformation of markers (more precisely the position of the camera
*  in function of the marker coordinate system).
*
*   \remark
*
*   History :
*
*  \author Hirokazu Kato kato@sys.im.hiroshima-cu.ac.jp
*  \version 3.1
*  \date 01/12/07
**/
/*  ----------------------------------------------------------------------
*   History :
*   Rev           Date              Who              Changes
*
*----------------------------------------------------------------------*/

#ifndef AR_H
#define AR_H
#ifdef __cplusplus
extern "C" {
#endif

//
//////////////////////////////////////////////////////////////////////////
//////
//         Public includes.
//
//////////////////////////////////////////////////////////////////////////
//////

#include <stdio.h>
#ifndef __APPLE__
#include <malloc.h>
#else
#include <stdlib.h>
#endif

#include <AR/config.h>
```

```
#include <AR/param.h>

//
//  Public types and defines.
//

/** \def arMalloc(V,T,S)
* \brief allocation macro function
*
* allocate S element of type T.
* \param V returned allocated area pointer
* \param T type of element
* \param S number of element
*/
#define arMalloc(V,T,S)  \
{ if( ((V) = (T *)malloc( sizeof(T) * (S) )) == 0 ) \
{printf("malloc error!!\n"); exit(1);} }

/* overhead ARToolkit type*/
typedef char             ARInt8;
typedef short            ARInt16;
typedef int              ARInt32;
typedef unsigned char    ARUint8;
typedef unsigned short   ARUint16;
typedef unsigned int     ARUint32;

/** \struct ARMarkerInfo
* \brief main structure for detected marker.
*
* Store information after contour detection (in idea screen coordinate, after distorsion
compensated).
* \remark lines are represented by 3 values a,b,c for ax+by+c=0
* \param area number of pixels in the labeled region
* \param id marker identitied number
* \param dir Direction that tells about the rotation about the marker (possible values are 0, 1, 2 or
3). This parameter makes it possible to tell about the line order of the detected marker (so which
line is the first one) and so find the first vertex. This is important to compute the transformation
matrix in arGetTransMat().
* \param cf confidence value (probability to be a marker)
* \param pos center of marker (in ideal screen coordinates)
* \param line line equations for four side of the marker (in ideal screen coordinates)
* \param vertex edge points of the marker (in ideal screen coordinates)
*/
typedef struct {
    int    area;
    int    id;
    int    dir;
    double  cf;
    double  pos[2];
```

```
    double  line[4][3];
    double  vertex[4][2];
} ARMarkerInfo;

/** \struct ARMarkerInfo2
* \brief internal structure use for marker detection.
*
* Store information after contour detection (in observed screen coordinate, before distorsion
correction).
* \param area number of pixels in the labeled region
* \param pos position of the center of the marker (in observed screen coordinates)
* \param coord_num numer of pixels in the contour.
* \param x_coord x coordinate of the pixels of contours (size limited by AR_CHAIN_MAX).
* \param y_coord y coordinate of the pixels of contours (size limited by AR_CHAIN_MAX).
* \param vertex position of the vertices of the marker. (in observed screen coordinates)
                    rem:the first vertex is stored again as the 5th entry in the array – for convenience of
drawing a line-strip easier.
*
*/
typedef struct {
    int     area;
    double  pos[2];
    int     coord_num;
    int     x_coord[AR_CHAIN_MAX];
    int     y_coord[AR_CHAIN_MAX];
    int     vertex[5];
} ARMarkerInfo2;

//
================================================================================
======
//        Public globals.
//
================================================================================
======

/** \var int arDebug
* \brief activate artoolkit debug mode
*
* control debug informations in ARToolKit.
* the possible values are:
* - 0: not in debug mode
* - 1: in debug mode
* by default: 0
*/
extern int      arDebug;

/** \var ARUint8 *arImage
* \brief internal image
*
* internal image used. (access only for debugging ARToolKit)
* by default: NULL
*/
```

```
extern ARUint8  *arImage;

/** \var int arFittingMode
* \brief fitting display mode use by ARToolkit.
*
* Correction mode for the distorsion of the camera.
* You can enable a correction with a texture mapping.
* the possible values are:
* - AR_FITTING_TO_INPUT: input image
* - AR_FITTING_TO_IDEAL: compensated image
* by default: DEFAULT_FITTING_MODE in config.h
*/
extern int      arFittingMode;

/** \var int arImageProcMode
* \brief define the image size mode for marker detection.
*
* Video image size for marker detection. This control
* if all the image is analyzed
* the possible values are :
* - AR_IMAGE_PROC_IN_FULL: full image uses.
* - AR_IMAGE_PROC_IN_HALF: half image uses.
* by default: DEFAULT_IMAGE_PROC_MODE in config.h
*/
extern int      arImageProcMode;

/** \var ARParam arParam
* \brief internal intrinsic camera parameter
*
* internal variable for camera intrinsic parameters
*/
extern ARParam  arParam;

/** \var int arImXsize
* \brief internal image size in width.
*
* internal image size in width (generally initialize in arInitCparam)
*/
/** \var int arImYsize
* \brief internal image size in heigth
*
* internal image size in heigth (generally initialize in arInitCparam)
*/
extern int      arImXsize, arImYsize;

/** \var int arTemplateMatchingMode
* \brief XXXBK
*
* XXXBK
* the possible values are :
* AR_TEMPLATE_MATCHING_COLOR: Color Template
* AR_TEMPLATE_MATCHING_BW: BW Template
* by default: DEFAULT_TEMPLATE_MATCHING_MODE in config.h
```

```
*/
extern int      arTemplateMatchingMode;

/** \var int arMatchingPCAMode
* \brief XXXBK
*
* XXXBK
* the possible values are :
* -AR_MATCHING_WITHOUT_PCA: without PCA
* -AR_MATCHING_WITH_PCA: with PCA
* by default: DEFAULT_MATCHING_PCA_MODE in config.h
*/
extern int      arMatchingPCAMode;

//
//////////////////////////////////////////////////////////////////////////////
//////
//      Public functions.
//
//////////////////////////////////////////////////////////////////////////////
//////

/*
  Initialization
*/

/** \fn int arInitCparam( ARParam *param )
* \brief initialize camera parameters.
*
* set the camera parameters specified in the camera parameters structure
* *param to static memory in the AR library. These camera parameters are
* typically read from a data file at program startup. In the video-see through
* AR applications, the default camera parameters are sufficient, no camera
* calibration is needed.
* \param param the camera parameter structure
* \return always 0
*/
int arInitCparam( ARParam *param );

/** \fn int arLoadPatt( char *filename )
* \brief load markers description from a file
*
* load the bitmap pattern specified in the file filename into the pattern
* matching array for later use by the marker detection routines.
* \param filename name of the file containing the pattern bitmap to be loaded
* \return the identity number of the pattern loaded or –1 if the pattern load failed.
*/
int arLoadPatt( const char *filename );

/*
  Detection
*/
```

```
/** \fn arDetectMarker( ARUint8 *dataPtr, int thresh, ARMarkerInfo **marker_info, int
*marker_num )
* \brief main function to detect the square markers in the video input frame.
*
* This function proceeds to thresholding, labeling, contour extraction and line corner estimation
* (and maintains an history).
* It's one of the main function of the detection routine with arGetTransMat.
* \param dataPtr a pointer to the color image which is to be searched for square markers.
*              The pixel format depend of your architecture. Generally ABGR, but the images
*              are treated as a gray scale, so the order of BGR components does not matter.
*              However the ordering of the alpha comp, A, is important.
* \param thresh  specifies the threshold value (between 0-255) to be used to convert
*              the input image into a binary image.
* \param marker_info a pointer to an array of ARMarkerInfo structures returned
*                which contain all the information about the detected squares in the image
* \param marker_num the number of detected markers in the image.
* \return 0 when the function completes normally, -1 otherwise
*/
int arDetectMarker( ARUint8 *dataPtr, int thresh,
                ARMarkerInfo **marker_info, int *marker_num );

/** \fn arDetectMarkerLite( ARUint8 *dataPtr, int thresh,
                  ARMarkerInfo **marker_info, int *marker_num )
* \brief main function to detect rapidly the square markers in the video input frame.
*
* this function is a simpler version of arDetectMarker that does not have the
* same error correction functions and so runs a little faster, but is more error prone
*
* \param dataPtr a pointer to the color image which is to be searched for square markers.
*              The pixel format depend of your architecture. Generally ABGR, but the images
*              are treated as a gray scale, so the order of BGR components does not matter.
*              However the ordering of the alpha component, A, is important.
* \param thresh  specifies the threshold value (between 0-255) to be used to convert
*              the input image into a binary image.
* \param marker_info a pointer to an array of ARMarkerInfo structures returned
*                which contain all the information about the detected squares in the image
* \param marker_num the number of detected markers in the image.
* \return 0 when the function completes normally, -1 otherwise
*/
int arDetectMarkerLite( ARUint8 *dataPtr, int thresh,
                  ARMarkerInfo **marker_info, int *marker_num );

/** \fn arGetTransMat( ARMarkerInfo *marker_info,
                double center[2], double width, double conv[3][4] )
* \brief compute camera position in function of detected markers.
*
* calculate the transformation between a detected marker and the real camera,
* i.e. the position and orientation of the camera relative to the tracking mark.
* \param marker_info the structure containing the parameters for the marker for
*                which the camera position and orientation is to be found relative to.
*                This structure is found using arDetectMarker.
* \param center the physical center of the marker. arGetTransMat assumes that the marker
*             is in x-y plane, and z axis is pointing downwards from marker plane.
```

```
*              So vertex positions can be represented in 2D coordinates by ignoring the
*              z axis information. The marker vertices are specified in order of clockwise.
* \param width the size of the marker (in mm).
* \param conv the transformation matrix from the marker coordinates to camera coordinate frame,
*             that is the relative position of real camera to the real marker
* \return always 0.
*/
double arGetTransMat( ARMarkerInfo *marker_info,
                      double center[2], double width, double conv[3][4] );

/** \fn arGetTransMatCont( ARMarkerInfo *marker_info, double prev_conv[3][4],
                       double center[2], double width, double conv[3][4] )
* \brief compute camera position in function of detected marker with an history function.
*
* calculate the transformation between a detected marker and the real camera,
* i.e. the position and orientation of the camera relative to the tracking mark. Since
* this routine operate on previous values, the result are more stable (less jittering).
*
* \param marker_info the structure containing the parameters for the marker for
*                    which the camera position and orientation is to be found relative to.
*                    This structure is found using arDetectMarker.
* \param prev_conv the previous transformation matrix obtain.
* \param center the physical center of the marker. arGetTransMat assumes that the marker
*               is in x-y plane, and z axis is pointing downwards from marker plane.
*               So vertex positions can be represented in 2D coordinates by ignoring the
*               z axis information. The marker vertices are specified in order of clockwise.
* \param width the size of the marker (in mm).
* \param conv the transformation matrix from the marker coordinates to camera coordinate frame,
*             that is the relative position of real camera to the real marker
* \return always 0.
*/
double arGetTransMatCont( ARMarkerInfo *marker_info, double prev_conv[3][4],
                      double center[2], double width, double conv[3][4] );

double arGetTransMat2( double rot[3][3], double pos2d[][2],
                       double pos3d[][2], int num, double conv[3][4] );
double arGetTransMat3( double rot[3][3], double ppos2d[][2],
                      double ppos3d[][2], int num, double conv[3][4],
                      double *dist_factor, double cpara[3][4] );
double arGetTransMat4( double rot[3][3], double ppos2d[][2],
                       double ppos3d[][3], int num, double conv[3][4] );
double arGetTransMat5( double rot[3][3], double ppos2d[][2],
                       double ppos3d[][3], int num, double conv[3][4],
                       double *dist_factor, double cpara[3][4] );

/** \fn int arFreePatt( int patt_no )
* \brief remove a pattern from memory.
*
* desactivate a pattern and remove from memory. post-condition
* of this function is unavailability of the pattern.
* \param patt_no number of pattern to free
* \return return 1 in success, -1 if error
*/
```

```
int arFreePatt( int patt_no );

/** \fn int arActivatePatt( int pat_no )
* \brief activate a pattern on the recognition procedure.
*
* Activate a pattern to be check during the template matching
* operation.
* \param patt_no number of pattern to activate
* \return return 1 in success, -1 if error
*/
int arActivatePatt( int pat_no );

/** \fn int arDeactivatePatt( int pat_no )
* \brief desactivate a pattern on the recognition procedure.
*
* Desactivate a pattern for not be check during the template matching
* operation.
* \param patt_no number of pattern to desactivate
* \return return 1 in success, -1 if error
*/
int arDeactivatePatt( int pat_no );

/** \fn int arSavePatt( ARUint8 *image,
            ARMarkerInfo *marker_info, char *filename )
* \brief save a marker.
*
* used in mk_patt to save a bitmap of the pattern of the currently detected marker.
* The saved image is a table of the normalized viewed pattern.
* \param image a pointer to the image containing the marker pattern to be trained.
* \param marker_info a pointer to the ARMarkerInfo structure of the pattern to be trained.
* \param filename The name of the file where the bitmap image is to be saved.
* \return 0 if the bitmap image is successfully saved, -1 otherwise.
*/
int arSavePatt( ARUint8 *image,
            ARMarkerInfo *marker_info, char *filename );


/*
   Utility
*/

/** \fn int arUtilMatInv( double s[3][4], double d[3][4] )
* \brief Inverse a non-square matrix.
*
* Inverse a matrix in a non homogeneous format. The matrix
* need to be euclidian.
* \param s matrix input
* \param d resulted inverse matrix.
* \return 0 if the inversion success, -1 otherwise
* \remark input matrix can be also output matrix
*/
int    arUtilMatInv( double s[3][4], double d[3][4] );
```

```
/** \fn int arUtilMatMul( double s1[3][4], double s2[3][4], double d[3][4] )
* \brief Multiplication of two matrix.
*
* This procedure do a multiplication matrix between s1 and s2 and return
* the result in d : d=s1*s2. The precondition is the output matrix
* need to be different of input matrix. The precondition is euclidian matrix.
* \param s1 first matrix.
* \param s2 second matrix.
* \param d resulted multiplication matrix.
* \return 0 if the multiplication success, -1 otherwise
*/
int    arUtilMatMul( double s1[3][4], double s2[3][4], double d[3][4] );

/** \fn int arUtilMat2QuatPos( double m[3][4], double q[4], double p[3] )
* \brief extract a quaternion/position of matrix.
*
* Extract a rotation (quaternion format) and a position (vector format)
* from a transformation matrix. The precondition is an euclidian matrix.
* \param m source matrix
* \param q a rotation represented by a quaternion.
* \param p a translation represented by a vector.
* \return 0 if the extraction success, -1 otherwise (quaternion not normalize)
*/
int    arUtilMat2QuatPos( double m[3][4], double q[4], double p[3] );

/** \fn int arUtilQuatPos2Mat( double q[4], double p[3], double m[3][4] )
* \brief create a matrix with a quaternion/position.
*
* Create a transformation matrix from a quaternion rotation and a vector translation.
* \param q a rotation represented by a quaternion.
* \param p a translation represented by a vector.
* \param m destination matrix
* \return always 0
*/
int    arUtilQuatPos2Mat( double q[4], double p[3], double m[3][4] );

/** \fn void arUtilTimer(void)
* \brief get the time with the ARToolkit timer.
*
* Give the time elapsed since the reset of the timer.
* \return elapsed time (in milliseconds)
*/
double arUtilTimer(void);

/** \fn void arUtilTimerReset(void)
* \brief reset the internal timer of ARToolkit.
*
* Reset the internal timer used by ARToolKit.
* timer measurement (with arUtilTimer()).
*/
void   arUtilTimerReset(void);

/** \fn void arUtilSleep( int msec )
```

```
* \brief sleep the actual thread.
*
* Sleep the actual thread.
* \param msec time to sleep (in millisecond)
*/
void   arUtilSleep( int msec );

/*
  Internal processing
*/

/** \fn ARInt16 *arLabeling( ARUint8 *image, int thresh,
                    int *label_num, int **area, double **pos, int **clip,
                    int **label_ref )
* \brief extracted connected component from image.
*
* Labeling the input image, i.e. extracted connected component from the
* inptu video image.
* \param image input image
* \param thresh lighting threshold
* \param label_num number of detected components
* \param area XXXBK
* \param pos XXXBK
* \param clip XXXBK
* \param label_ref XXXBK
* \return XXXBK
*/
ARInt16 *arLabeling( ARUint8 *image, int thresh,
                    int *label_num, int **area, double **pos, int **clip,
                    int **label_ref );

/** \fn void arGetImgFeature( int *num, int **area, int **clip, double **pos )
* \brief  XXXBK
*
*  XXXBK
* \param num XXXBK
* \param area XXXBK
* \param clip XXXBK
* \param pos XXXBK
*/
void arGetImgFeature( int *num, int **area, int **clip, double **pos );

/** \fn ARMarkerInfo2 *arDetectMarker2( ARInt16 *limage,
                            int label_num, int *label_ref,
                            int *warea, double *wpos, int *wclip,
                            int area_max, int area_min, double factor, int *marker_num )
* \brief   XXXBK
*
*   XXXBK
* \param limage  XXXBK
* \param label_num  XXXBK
* \param label_ref  XXXBK
* \param warea  XXXBK
```

```
* \param wpos  XXXBK
* \param wclip  XXXBK
* \param area_max  XXXBK
* \param area_min  XXXBK
* \param factor  XXXBK
* \param marker_num  XXXBK
* \return XXXBK  XXXBK
*/
ARMarkerInfo2 *arDetectMarker2( ARInt16 *limage,
                        int label_num, int *label_ref,
                        int *warea, double *wpos, int *wclip,
                        int area_max, int area_min, double factor, int *marker_num );

/** \fn ARMarkerInfo *arGetMarkerInfo( ARUint8 *image,
                       ARMarkerInfo2 *marker_info2, int *marker_num )
* \brief information on
*
*  XXXBK
* \param image XXXBK
* \param marker_info2 XXXBK
* \param marker_num XXXBK
* \return XXXBK
*/
ARMarkerInfo *arGetMarkerInfo( ARUint8 *image,
                       ARMarkerInfo2 *marker_info2, int *marker_num );

/** \fn int arGetCode( ARUint8 *image, int *x_coord, int *y_coord, int *vertex,
            int *code, int *dir, double *cf )
* \brief  XXXBK
*
*  XXXBK
* \param image XXXBK
* \param x_coord XXXBK
* \param y_coord XXXBK
* \param vertex XXXBK
* \param code XXXBK
* \param dir XXXBK
* \param cf XXXBK
* \return XXXBK
*/
int arGetCode( ARUint8 *image, int *x_coord, int *y_coord, int *vertex,
            int *code, int *dir, double *cf );

/** \fn int arGetPatt( ARUint8 *image, int *x_coord, int *y_coord, int *vertex,
            ARUint8 ext_pat[AR_PATT_SIZE_Y][AR_PATT_SIZE_X][3] )
* \brief return a normalized pattern from a video image.
*
* This function return a normalized pattern from a video image. The
* format is a table with AR_PATT_SIZE_X by AR_PATT_SIZE_Y
* \param image video input image
* \param x_coord XXXBK
* \param y_coord XXXBK
* \param vertex XXXBK
```

```
* \param ext_pat detected pattern.
* \return  XXXBK
*/
int arGetPatt( ARUint8 *image, int *x_coord, int *y_coord, int *vertex,
               ARUint8 ext_pat[AR_PATT_SIZE_Y][AR_PATT_SIZE_X][3] );

/** \fn int arGetLine(int x_coord[], int y_coord[], int coord_num,
            int vertex[], double line[4][3], double v[4][2])
* \brief estimate a line from a list of point.
*
* Compute a linear regression from a list of point.
* \param x_coord X coordinate of points
* \param y_coord Y coordinate of points
* \param coord_num number of points
* \param vertex XXXBK
* \param line XXXBK
* \param v XXXBK
* \return  XXXBK
*/
int arGetLine(int x_coord[], int y_coord[], int coord_num,
            int vertex[], double line[4][3], double v[4][2]);

/** \fn int arGetContour( ARInt16 *limage, int *label_ref,
                int label, int clip[4], ARMarkerInfo2 *marker_info2 )
* \brief  XXXBK
*
*  XXXBK
* \param limage XXXBK
* \param label_ref XXXBK
* \param label XXXBK
* \param clip XXXBK
* \param marker_info2 XXXBK
* \return  XXXBK
*/
int arGetContour( ARInt16 *limage, int *label_ref,
                int label, int clip[4], ARMarkerInfo2 *marker_info2 );

/** \fn double arModifyMatrix( double rot[3][3], double trans[3], double cpara[3][4],
                        double vertex[][3], double pos2d[][2], int num )
* \brief  XXXBK
*
*  XXXBK
* \param rot XXXBK
* \param trans XXXBK
* \param cpara XXXBK
* \param vertex XXXBK
* \param pos2d XXXBK
* \param num XXXBK
* \return XXXBK
*/
double arModifyMatrix( double rot[3][3], double trans[3], double cpara[3][4],
                        double vertex[][3], double pos2d[][2], int num );
```

```
/** \fn int arGetAngle( double rot[3][3], double *wa, double *wb, double *wc )
* \brief extract euler angle from a rotation matrix.
*
* Based on a matrix rotation representation, furnish the cprresponding euler angles.
* \param rot the initial rotation matrix
* \param wa XXXBK:which element ?
* \param wb XXXBK:which element ?
* \param wc XXXBK:which element ?
* \return XXXBK
*/
int arGetAngle( double rot[3][3], double *wa, double *wb, double *wc );

/** \fn int arGetRot( double a, double b, double c, double rot[3][3] )
* \brief create a rotation matrix with euler angle.
*
* Based on a euler description, furnish a rotation matrix.
* \param a XXXBK:which element ?
* \param b XXXBK:which element ?
* \param c XXXBK:which element ?
* \param rot the resulted rotation matrix
* \return XXXBK
*/
int arGetRot( double a, double b, double c, double rot[3][3] );

/** \fn int arGetNewMatrix( double a, double b, double c,
                double trans[3], double trans2[3][4],
                double cpara[3][4], double ret[3][4] )
* \brief XXXBK
*
* XXXBK
* \param a XXXBK
* \param b XXXBK
* \param c XXXBK
* \param trans XXXBK
* \param trans2 XXXBK
* \param cpara XXXBK
* \param ret XXXBK
* \return XXXBK
*/
int arGetNewMatrix( double a, double b, double c,
                double trans[3], double trans2[3][4],
                double cpara[3][4], double ret[3][4] );

/** \fn int arGetInitRot( ARMarkerInfo *marker_info, double cpara[3][4], double rot[3][3] )
* \brief XXXBK
*
* XXXBK:initial of what ?
* \param marker_info XXXBK
* \param cpara XXXBK
* \param rot XXXBK
* \return XXXBK
*/
int arGetInitRot( ARMarkerInfo *marker_info, double cpara[3][4], double rot[3][3] );
```

```
/** \struct arPrevInfo
* \brief structure for temporal continuity of tracking
*
* History structure for arDetectMarkerLite and arGetTransMatCont
*/
typedef struct {
    ARMarkerInfo  marker;
    int     count;
} arPrevInfo;


/*-----------------------------------*/

extern ARUint8  *arImageL;
extern ARUint8  *arImageR;
extern ARSParam arsParam;
extern double   arsMatR2L[3][4];

int          arsInitCparam      ( ARSParam *sparam );
void         arsGetImgFeature   ( int *num, int **area, int **clip, double **pos, int LorR );
ARInt16      *arsLabeling       ( ARUint8 *image, int thresh,
                                  int *label_num, int **area, double **pos, int **clip,
                                  int **label_ref, int LorR );
int          arsGetLine         ( int x_coord[], int y_coord[], int coord_num,
                                  int vertex[], double line[4][3], double v[4][2], int LorR);
ARMarkerInfo *arsGetMarkerInfo   ( ARUint8 *image,
                                  ARMarkerInfo2 *marker_info2, int *marker_num, int LorR );
int          arsDetectMarker    ( ARUint8 *dataPtr, int thresh,
                                  ARMarkerInfo **marker_info, int *marker_num, int LorR );
int          arsDetectMarkerLite( ARUint8 *dataPtr, int thresh,
                                  ARMarkerInfo **marker_info, int *marker_num, int LorR );
double       arsGetTransMat     ( ARMarkerInfo *marker_infoL, ARMarkerInfo *marker_infoR,
                                  double center[2], double width,
                                  double transL[3][4], double transR[3][4] );
double       arsGetTransMatCont ( ARMarkerInfo *marker_infoL, ARMarkerInfo *marker_infoR,
                                  double prev_conv[3][4],
                                  double center[2], double width,
                                  double transL[3][4], double transR[3][4] );
double       arsGetTransMat2    ( double rot[3][3],
                                  double ppos2dL[][2], double ppos3dL[][3], int numL,
                                  double ppos2dR[][2], double ppos3dR[][3], int numR,
                                  double transL[3][4], double transR[3][4] );
double       arsGetPosErr( double pos2dL[2], double pos2dR[2] );
int          arsCheckPosition   ( double pos2dL[2], double pos2dR[2], double thresh );
int          arsCheckMarkerPosition( ARMarkerInfo *marker_infoL, ARMarkerInfo *marker_infoR,
                                     double thresh );

double arsModifyMatrix( double rot[3][3], double trans[3], ARSParam *arsParam,
                        double pos3dL[][3], double pos2dL[][2], int numL,
                        double pos3dR[][3], double pos2dR[][2], int numR );
ARMarkerInfo2 *arDetectMarker3( ARInt16 *limage, int label_num, int *label_ref,
                                int *warea, double *wpos, int *wclip,
```

```
                                                    int area_max, int area_min, int
*marker_num );
ARInt16 *arLabelingHSB( ARUint8 *image, int thresh,
                    int *label_num, int **area, double **pos, int **clip,
                    int **label_ref, int LorR, int minH, int maxH, int minS, int maxS, int minB, int
maxB, int invert);
ARInt16 *arLabelingHSBMultiple( ARUint8 *image, int thresh,
                    int *label_num, int **area, double **pos, int **clip,
                    int **label_ref, int LorR, int *minH, int *maxH, int *minS, int *maxS, int *minB,
int *maxB, int numHSV, int invert);
void RGBtoHSV(double r, double g, double b, double *H, double *S, double *V);
void HSVtoRGB(double h, double s, double v, double *R, double *G, double *B);

#ifdef __cplusplus
}
#endif
#endif
```

Fichero formatoPixel.h

```
#if defined(AR_PIX_FORMAT_ARGB)
#define B_COMP *(pnt+3)
#define G_COMP *(pnt+2)
#define R_COMP *(pnt+1)
#define A_COMP *(pnt+0)
#define VALORES_POR_PIXEL 4
#elif defined(AR_PIX_FORMAT_ABGR)
#define B_COMP *(pnt+1)
#define G_COMP *(pnt+2)
#define R_COMP *(pnt+3)
#define A_COMP *(pnt+0)
#define VALORES_POR_PIXEL 4
#elif defined(AR_PIX_FORMAT_BGRA)
#define B_COMP *(pnt+0)
#define G_COMP *(pnt+1)
#define R_COMP *(pnt+2)
#define A_COMP *(pnt+3)
#define VALORES_POR_PIXEL 4
#elif defined(AR_PIX_FORMAT_BGR)
#define B_COMP *(pnt+0)
#define G_COMP *(pnt+1)
#define R_COMP *(pnt+2)
#define VALORES_POR_PIXEL 3
#elif defined(AR_PIX_FORMAT_RGBA)
#define B_COMP *(pnt+2)
#define G_COMP *(pnt+1)
#define R_COMP *(pnt+0)
#define A_COMP *(pnt+3)
#define VALORES_POR_PIXEL 4
#elif defined(AR_PIX_FORMAT_RGB)
#define B_COMP *(pnt+2)
#define G_COMP *(pnt+1)
#define R_COMP *(pnt+0)
#define VALORES_POR_PIXEL 3
#else
#error formato de video no soportado
#endif
```

Fichero configuration.c

```
#ifdef _WIN32
#  include <windows.h>
#endif
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>


#ifndef __APPLE__
#  include <GL/glut.h>
#else
#  include <GLUT/glut.h>
#endif
#include <AR/gsub.h>
#include <AR/ar.h>
#include <AR/video.h>
#include <formatopixel.h>



#define MIN_H 0
#define MAX_H 0
#define MIN_S 0
#define MAX_S 0
#define MIN_B 0
#define MAX_B 0
#define UMBRAL 5


int           xsize, ysize;
int                                   thresh = 100;
int           count = 0;

int           minH = MIN_H, maxH = MAX_H, minS = MIN_S, maxS = MAX_S, minB = MIN_B,
maxB = MAX_B, umbral = UMBRAL;
int invert = 0;
ARUint8         *dataPtr;


/* set video capture configuration */
#ifdef _WIN32
char                        *vconf = "flipV,showDlg"; // see video.h for a list of supported parameters
#else
char                        vconf[256];
#endif

char          *cparam_name    = "Data/camera_para.dat";
ARParam         cparam;

static void   init(void);
```

```
static void   cleanup(void);
static void   keyEvent( unsigned char key, int x, int y);
static void   mouseEvent(int button, int state, int x, int y);
static void   mainLoop(void);
static int    guardarEnFichero(void);
static int    cargarDeFichero(void);


int main(int argc, char** args)
{

#ifndef _WIN32
char dev[256];
strcpy(vconf,"-dev=camara -channel=0 -palette=YUV420P -width=320 -height=240");


if(argc==2) {
 strcpy(vconf,"-channel=0 -palette=YUV420P -width=320 -height=240 -dev=");
 strcat(vconf, args[1]);
}
else {
 FILE *f;
 f=fopen("camara","r");
 if(f == NULL) {
  printf("No se ha encontrado un enlace \"./camara\" al dispositivo de video. Por favor, indique el
nombre de dispositivo (normalmente /dev/video0 o /dev/video1) en el que se encuentra su camara:
");
  scanf("%s",&dev);
  strcpy(vconf, "-channel=0 -palette=YUV420P -width=320 -height=240 -dev=");
  strcat(vconf, dev);


 }
 else {
 fclose(f);
 }

}
#endif
        //initialize applications
        printf("\nEste programa sirve para entrenar al sistema a reconocer el objetivo a seguir:");
   printf("\n\n\n\t1.Situe el objetivo delante de la camara.");
        printf("\n\n\t2.Cuando vea su objetivo en la pantalla, haga click sobre el\n\tcon el boton
derecho del raton.");
        printf("\n\n\t3.Realice varios clicks sobre la superficie del objetivo a seguir\n\tcon el boton
izquierdo del raton, hasta que la superficie de este\n\taparezca totalmente bordeada.");
        printf("\n\n\t4.Tambiï¿½ puede ajustar manualmente los rangos de H, S y V que\n\tse
detectaran,pulsando respectivamente las teclas 'h', 's' y 'b'.");
        printf("\n\n\t5.Puede ajustar un umbral pulsando 'u'. Cuanto mayor sea el umbral,\n\tmayor
sera la desviacion permitida respecto de las componentes\n\tHSV elegidas.");
        printf("\n\n\t6.Cuando termine pulse 'g' para guardar la configuracion en el\n\tfichero
conf.con.\n\n\n");
```

```
    init();

        //start video capture
        arVideoCapStart();

        //start the main event loop
    argMainLoop( mouseEvent, keyEvent, mainLoop );

        return 0;
}

static void   keyEvent( unsigned char key, int x, int y)
{    char resp;
    /* quit if the ESC key is pressed */
    if( key == 0x1b ) {
       cleanup();
       exit(0);
    }

        if( key == 'h' ) {
                printf("\nminH = %d, maxH = %d\n",minH,maxH);
           printf("\nIntroduce el valor minimo de H(tono):\n");
                scanf("%d",&minH);
           printf("\nIntroduce el valor maximo de H(tono):\n");
                scanf("%d",&maxH);

        }

    if( key == 'b' ) {
           printf("\nminB = %d, maxB = %d\n",minB,maxB);
           printf("\nIntroduce el valor minimo de B(brillo):\n");
                scanf("%d",&minB);
           printf("\nIntroduce el valor maximo de B(brillo):\n");
                scanf("%d",&maxB);
        }

    if( key == 's' ) {
                printf("\nminS = %d, maxS = %d\n",minS,maxS);
           printf("\nIntroduce el valor minimo de S(Saturacion):\n");
                scanf("%d",&minS);
           printf("\nIntroduce el valor maximo de S(Saturacion):\n");
                scanf("%d",&maxS);

        }

         if( key == 'i' ) {

                 if(invert) invert = 0;
                  else invert = 1;
            printf("\nInversion realizada\n");

        }
```

```
        if( key == 'u' ) {

            printf("\nIntroduce el umbral:\n");
                scanf("%d",&umbral);

        }

    if( key == 'g' ) {

            printf("\nValores actuales: \nH: [%d-%d]\nS: [%d-%d]\nB: [%d-%d]
\n",minH,maxH,minS,maxS,minB,maxB);
            printf("\nDesea guardar los datos de configuracion de la sesion actual? (s/n)\n");
                scanf("%c",&resp);
                if(resp=='s') {
        if(guardarEnFichero()) printf("\nConfiguracion guardada en fichero.\n");
                 else {printf("\nError!. No se pudo guardar la configuracion en el fichero.\n");}
                }


        }

}

static void mouseEvent(int button, int state, int x, int y) {
  double H, S, V;
  ARUint8 *pnt;
  x=x/2;
  y=y/2;

  //varios clicks para autoajustar los valores H, S y V

if (button == GLUT_LEFT_BUTTON && state == GLUT_DOWN) {
pnt = (dataPtr+(arImXsize*VALORES_POR_PIXEL*(y-1))+(x*VALORES_POR_PIXEL));
                RGBtoHSV(R_COMP, G_COMP, B_COMP, &H, &S, &V);

                if(H>=(maxH-umbral)) {maxH=(H+umbral);}
                if(H<=(minH+umbral)) {minH=(H-umbral);}
              if(S>=(maxS-umbral)) {maxS=(S+umbral);}
                if(S<=(minS+umbral)) {minS=(S+umbral);}
                if(V>=(maxB-umbral)) {maxB=(V+umbral);}
                if(V<=(minB+umbral)) {minB=(V-umbral);}

                printf("\nValores del pixel:\n%f %f %f\n",H, S, V);
                printf("\nNuevos rangos: \nH: [%d-%d]\nS: [%d-%d]\nB: [%d-%d]
\n",minH,maxH,minS,maxS,minB,maxB);

    }

  //para declarar el punto inicial del objetivo
  if (button == GLUT_RIGHT_BUTTON && state == GLUT_DOWN) {
pnt = (dataPtr+(arImXsize*VALORES_POR_PIXEL*(y-1))+(x*VALORES_POR_PIXEL));

        RGBtoHSV(R_COMP, G_COMP, B_COMP, &H, &S, &V);
```

```
            minH=H-umbral;
                maxH=H+umbral;
                minS=S-umbral;
                maxS=S+umbral;
                minB=S-umbral;
                maxB=S+umbral;
                printf("\nValores del pixel:\n%f %f %f\n",H, S, V);


    }

}



/* main loop */
static void mainLoop(void)
{

//    ARMarkerInfo    *marker_info;
          ARMarkerInfo2   *marker2;
          int             marker_num2;
    //int           marker_num;
    int           i,j;
          ARInt16         *limage;
    int           label_num;
    int           *area, *clip, *label_ref;
    double          *pos;



    marker_num2 = 0;


    /* grab a video frame */
    if( (dataPtr = (ARUint8 *)arVideoGetImage()) == NULL ) {
        arUtilSleep(2);
        return;
    }

    if( count == 0 ) arUtilTimerReset();
    count++;

          /*draw the video*/
    argDrawMode2D();
    argDispImage( dataPtr, 0,0 );

          /* capture the next video frame */
          arVideoCapNext();

          glColor3f( 1.0, 0.0, 0.0 );
```

```
        glLineWidth(6.0);


   limage = arLabelingHSB( dataPtr, thresh,
                  &label_num, &area, &pos, &clip, &label_ref, 1, minH, maxH, minS, maxS, minB,
maxB, invert);
   if( limage == 0 )   {
           cleanup();
           exit(0);

        }

   marker2 = arDetectMarker3( limage, label_num, label_ref,
                        area, pos, clip, AR_AREA_MAX, AR_AREA_MIN, &marker_num2);
   if( marker2 == 0 ) {
           cleanup();
           exit(0);

        }
        argDrawMode2D();
        glColor3f( 1.0, 0.0, 0.0 );

        for( i = 0; i < marker_num2; i++ ) {
                for(j=0; j<marker2[i].coord_num-1; j++) {
                 argLineSeg( marker2[i].x_coord[j] ,  marker2[i].y_coord[j],  marker2[i].x_coord
[j+1],  marker2[i].y_coord[j+1], 0, 0);
                }
        }
        /*swap the graphics buffers*/
        argSwapBuffers();
}

static void init( void )
{
        ARParam  wparam;
   cargarDeFichero();
   /* open the video path */
   if( arVideoOpen( vconf ) < 0 ) exit(0);
   /* find the size of the window */
   if( arVideoInqSize(&xsize, &ysize) < 0 ) exit(0);
   printf("Image size (x,y) = (%d,%d)\n", xsize, ysize);

   /* set the initial camera parameters */
   if( arParamLoad(cparam_name, 1, &wparam) < 0 ) {
      printf("Camera parameter load error !!\n");
      exit(0);
   }
   arParamChangeSize( &wparam, xsize, ysize, &cparam );
   arInitCparam( &cparam );
   printf("*** Camera Parameter ***\n");
   arParamDisp( &cparam );
```

```
    /* open the graphics window */
    argInit( &cparam, 2.0, 0, 0, 0, 0 );
}

/* cleanup function called when program exits */
static void cleanup(void)
{
          arVideoCapStop();
    arVideoClose();
    argCleanup();
}

/* draw the the AR objects */



static int cargarDeFichero(void) {
 FILE *fich;
 char c;
 fich=fopen("conf.con","r");

 if(fich != NULL) {
           fscanf(fich,"%d",&umbral);
           fscanf(fich," %c",&c);

           if(c=='#'){  //si el separador es correcto y no es fin de fichero carga valores;
            fscanf(fich,"%d",&minH);
            fscanf(fich,"%d",&maxH);
            fscanf(fich,"%d",&minS);
            fscanf(fich,"%d",&maxS);
            fscanf(fich,"%d",&minB);
            fscanf(fich,"%d",&maxB);
           }
  fclose(fich);
  return 1;
 }

 return 0;

}

static int guardarEnFichero(void) {
 FILE *fich;

 fich=fopen("conf.con","w");

 if(fich != NULL) {

  fprintf(fich,"%d\n#\n%d %d\n%d %d\n%d %d\n-
\n",umbral,minH,maxH,minS,maxS,minB,maxB);
  fclose(fich);
  return 1;
```

```
 }
 return 0;

}
```

Fichero sample2.c

```
#include <allegro.h>
#ifdef _WIN32
#  include <winalleg.h>
#endif
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>

#ifndef __APPLE__
#  include <GL/glut.h>
#else
#  include <GLUT/glut.h>
#endif
#include <AR/gsub.h>
#include <AR/param.h>
#include <AR/ar.h>
#include <AR/video.h>
#include <time.h>
#include <formatoPixel.h>

#define UMBRAL 5

AUDIOSTREAM *stream;
unsigned char *p;
int offset=0;
int timer=0;

int          xsize, ysize;
int                              thresh = 100;
int                              sonando=0;

int          minH[10], maxH[10], minS[10], maxS[10], minB[10], maxB[10], numHSV=0, umbral
= UMBRAL;

int invert = 0;
ARUint8         *dataPtr;


/* set video capture configuration */
#ifdef _WIN32
char                     *vconf = "flipV,flipH,showDlg"; // see video.h for a list of supported
parameters
#else
char                     vconf[256];
#endif

char          *cparam_name    = "Data/camera_para.dat";
ARParam         cparam;

static void   cleanup(void);
```

```
static void   init(void);
static void   keyEvent( unsigned char key, int x, int y);
static int    generate_audio(unsigned char tono);
static void   mainLoop(void);
static int    guardarEnFichero(void);
static int    cargarDeFichero(void);


int main(int argc, char** args)
{
        //initialize applications

#ifndef _WIN32
char dev[256];
strcpy(vconf,"-dev=camara -channel=0 -width=320 -height=240 -palette=YUV420P");


if(argc==2) {
 strcpy(vconf,"-channel=0 -palette=YUV420P -width=320 -height=240 -dev=");
 strcat(vconf, args[1]);
}
else {
 FILE *f;
 f=fopen("camara","r");
 if(f == NULL) {
  printf("No se ha encontrado un enlace \"./camara\" al dispositivo de video. Por favor, indique el
nombre de dispositivo (normalmente /dev/video0 o /dev/video1) en el que se encuentra su camara:
");
  scanf("%s",&dev);
  strcpy(vconf, "-channel=0 -palette=YUV420P -width=320 -height=240 -dev=");
  strcat(vconf, dev);


 }
 else {
 fclose(f);
 }

}
#endif

   init();

        //start video capture
        arVideoCapStart();

        //start the main event loop
   argMainLoop( NULL, keyEvent, mainLoop );

        return 0;
}
END_OF_MAIN();
```

```
static void   keyEvent( unsigned char key, int x, int y)
{    char resp;
   /* quit if the ESC key is pressed */
   if( key == 0x1b ) {
      cleanup();
      exit(0);
   }

         if( key == 'h' ) {
                  printf("\nminH = %d, maxH = %d\n",minH,maxH);
              printf("\nIntroduce el valor minimo de H(tono):\n");
                  scanf("%d",&minH);
              printf("\nIntroduce el valor maximo de H(tono):\n");
                  scanf("%d",&maxH);

         }

   if( key == 'b' ) {
              printf("\nminB = %d, maxB = %d\n",minB,maxB);
              printf("\nIntroduce el valor minimo de B(brillo):\n");
                  scanf("%d",&minB);
              printf("\nIntroduce el valor maximo de B(brillo):\n");
                  scanf("%d",&maxB);
         }

   if( key == 's' ) {
                  printf("\nminS = %d, maxS = %d\n",minS,maxS);
              printf("\nIntroduce el valor minimo de S(Saturacion):\n");
                  scanf("%d",&minS);
              printf("\nIntroduce el valor maximo de S(Saturacion):\n");
                  scanf("%d",&maxS);

         }



          if( key == 'i' ) {

                   if(invert) invert = 0;
                    else invert = 1;
               printf("\nInversion realizada\n");

         }

         if( key == 'u' ) {

              printf("\nIntroduce el umbral:\n");
                  scanf("%d",&umbral);

         }

   if( key == 'g' ) {
```

```
            printf("\nValores actuales: \nH: [%d-%d]\nS: [%d-%d]\nB: [%d-%d]\n",minH[0],maxH
[0],minS[0],maxS[0],minB[0],maxB[0]);
            printf("\nDesea guardar los datos de configuracion de la sesion actual? (s/n)\n");
                  scanf("%c",&resp);
                  if(resp=='s') {
        if(guardarEnFichero()) printf("\nConfiguracion guardada en fichero.\n");
                   else {printf("\nError!. No se pudo guardar la configuracion en el fichero.\n");}
                  }


         }
}


/* main loop */
static void mainLoop(void)
{

//   ARMarkerInfo    *marker_info;
         ARMarkerInfo2   *marker2;
         int             marker_num2;
   //int           marker_num;
   int           i,j, k, l, m;
         ARInt16         *limage;
   int           label_num;
   int           *area, *clip, *label_ref;
   double          *pos;
         double          H,S,V;
         int           colision=0;
         int           punto[2];
         ARUint8         *pnt;



   marker_num2 = 0;

   /* grab a video frame */
   if( (dataPtr = (ARUint8 *)arVideoGetImage()) == NULL ) {
      arUtilSleep(2);
      return;
   }

         /*draw the video*/
   argDrawMode2D();
   argDispImage( dataPtr, 0,0 );

         /* capture the next video frame */
         arVideoCapNext();

         glColor3f( 1.0, 0.0, 0.0 );
         glLineWidth(6.0);

         /* detect the markers in the video frame */
```

```
//        if(arDetectMarker(dataPtr, thresh,
//                &marker_info, &marker_num, &marker2) < 0 ) {

//           cleanup();
//                exit(0);
//        }



        limage = arLabelingHSBMultiple( dataPtr, thresh,
                 &label_num, &area, &pos, &clip, &label_ref, 1, minH, maxH, minS, maxS, minB,
maxB,numHSV, invert);

        if( limage == 0)   {
          cleanup();
          exit(0);

        }
   marker2 = arDetectMarker3( limage, label_num, label_ref,
                        area, pos, clip, AR_AREA_MAX, AR_AREA_MIN, &marker_num2);
   if( marker2 == 0) {
          cleanup();
          exit(0);

        }



        for( i = 0; i < marker_num2; i++ ) {
                for(j=0; j<marker2[i].coord_num-1; j++) {
                 argLineSeg( marker2[i].x_coord[j] ,  marker2[i].y_coord[j],  marker2[i].x_coord
[j+1],  marker2[i].y_coord[j+1], 0, 0);
                }
            //comprobamos de que color es el primer punto, asi sabemos si es el puntero (los valores
de color del puntero estan en la primera posicion del array, y del fichero de configuracion)
            pnt = (dataPtr+(arImXsize*VALORES_POR_PIXEL*(marker2[i].y_coord[0]-1))+
(marker2[i].x_coord[0]*VALORES_POR_PIXEL));
                RGBtoHSV(R_COMP,G_COMP,B_COMP, &H, &S, &V);
      if((H<maxH[0] && H>minH[0]) && (S<maxS[0] && S>minS[0]) && (V<maxB[0] &&
V>minB[0])) {
                  //para cada marca
                       for( k = 0; k < marker_num2; k++ ) {
                        pnt =(dataPtr+(arImXsize*VALORES_POR_PIXEL*(marker2[k].y_coord
[0]-1))+(marker2[k].x_coord[0]*VALORES_POR_PIXEL));
                        RGBtoHSV(R_COMP,G_COMP,B_COMP, &H, &S, &V);
                        //si no es del color del puntero (no es el puntero)
                        if(!((H<maxH[0] && H>minH[0]) && (S<maxS[0] && S>minS[0]) &&
(V<maxB[0] && V>minB[0]))){
                         colision=0;
                         //para cada punto de esta marca
                         for(l=0; l<marker2[k].coord_num-1; l++) {
                                 //comprobar si tiene algun punto cercano a algun punto del puntero,
```

```
en tal caso hay colision
                                    for(m=0;m<marker2[i].coord_num-1;m++) {
                                     if(abs((int)(marker2[i].x_coord[m]-marker2[k].x_coord[l]))<=10
&& abs((int)(marker2[i].y_coord[m]-marker2[k].y_coord[l]))<=10){
                                colision=1;
                                        punto[0]=marker2[k].x_coord[l];
                                        punto[1]=marker2[k].y_coord[l];
                                        break;
                                       }//end_if


                                        }//end_for
                                }//end_for

                                //si hubo colision
                                if(colision) {
                                       //Podemos comprobar el color del primer punto para saber de que
objeto se trata
                               // RGBtoHSV(*(dataPtr+(arImXsize*4*(marker2[k].y_coord[0]-1))+
(marker2[k].x_coord[0]*4)+2),*(dataPtr+(arImXsize*4*(marker2[k].y_coord[0]-1))+(marker2[k].
x_coord[0]*4)+1),*(dataPtr+(arImXsize*4*(marker2[k].y_coord[0]-1))+(marker2[k].x_coord[0]
*4)), &H, &S, &V);

                                        //Pero mejor si comprobamos el color del punto de colision
                                       pnt = (dataPtr+(arImXsize*VALORES_POR_PIXEL*(punto[1]-1))+
(punto[0]*VALORES_POR_PIXEL));
                                        RGBtoHSV(R_COMP,G_COMP,B_COMP, &H, &S, &V);
                                        if(!sonando) {
                                        if(numHSV>1 && (H<maxH[1] && H>minH[1]) && (S<maxS[1]
&& S>minS[1]) && (V<maxB[1] && V>minB[1])) {generate_audio(0x03);sonando=1;}
                                         else if(numHSV>2 && (H<maxH[2] && H>minH[2]) &&
(S<maxS[2] && S>minS[2]) && (V<maxB[2] && V>minB[2])) {generate_audio(0x04);
sonando=1;}
                              else if(numHSV>3 && (H<maxH[3] && H>minH[3]) && (S<maxS[3] &&
S>minS[3]) && (V<maxB[3] && V>minB[3])) {generate_audio(0x05);sonando=1;}
                                         else {generate_audio(0x06);sonando=1;}
                                        }//end_if
                                }//end_if
                               }//end_if
                        }//end_for
                       }//end_if
                }//end_for



        /*swap the graphics buffers*/

if(sonando && time(NULL)-timer>=1)  {
    stop_audio_stream(stream);
          timer=0;
          sonando=0;
          allegro_exit();
}
```

```
        argSwapBuffers();
}

static void init( void )
{
        ARParam  wparam;


        if(!cargarDeFichero()) { minH[0]=0;maxH[0]=0;minS[0]=0;maxS[0]=0;minB[0]=0;maxB
[0]=0;}
        /* selecciona el color de borrado */
   glClearColor (0.0, 0.0, 0.0, 0.0);
   /* open the video path */
   if( arVideoOpen( vconf ) < 0 ) exit(0);
   /* find the size of the window */
   if( arVideoInqSize(&xsize, &ysize) < 0 ) exit(0);
   printf("Image size (x,y) = (%d,%d)\n", xsize, ysize);

   /* set the initial camera parameters */
   if( arParamLoad(cparam_name, 1, &wparam) < 0 ) {
      printf("Camera parameter load error !!\n");
      exit(0);
   }
   arParamChangeSize( &wparam, xsize, ysize, &cparam );
   arInitCparam( &cparam );
   printf("*** Camera Parameter ***\n");
   arParamDisp( &cparam );



   /* open the graphics window */
   argInit( &cparam, 2.0, 0, 0, 0, 0 );
}

/* cleanup function called when program exits */
static void cleanup(void)
{
        arVideoCapStop();
   arVideoClose();
   argCleanup();
}

/* draw the the AR objects */



static int cargarDeFichero(void) {
 FILE *fich;
 char c;
 int i=0;

 fich=fopen("conf2.con","r");
```

```
 if(fich != NULL) {
          fscanf(fich,"%d",&umbral);
          fscanf(fich," %c",&c);

          if(c=='#')  //si el separador es correcto y no es fin de fichero inicia bucle
          for(i=0;i<4;i++){

           fscanf(fich,"%d",&minH[i]);
           fscanf(fich,"%d",&maxH[i]);
           fscanf(fich,"%d",&minS[i]);
           fscanf(fich,"%d",&maxS[i]);
           fscanf(fich,"%d",&minB[i]);
           fscanf(fich,"%d",&maxB[i]);
           fscanf(fich," %c",&c);
           if(c!='#') break; //si es el fin del fichero, o el separador no es correcto, finaliza el bucle

          }
  numHSV=i+1;
  fclose(fich);
  return 1;
 }

 return 0;

}

static int guardarEnFichero(void) {
 FILE *fich;
 int i;

 fich=fopen("conf2.con","w");

 if(fich != NULL) {
  fprintf(fich,"%d\n",umbral);
  for(i=0;i<numHSV;i++) fprintf(fich,"#\n%d %d\n%d %d\n%d %d\n",minH[i],maxH[i],minS[i],
maxS[i],minB[i],maxB[i]);
  fprintf(fich,"-");
  fclose(fich);
  return 1;
 }
 return 0;

}


static int generate_audio(unsigned char tono) {


  unsigned char incremento = tono;
  unsigned char valor=0x00;

     allegro_init();
```

```
        install_timer();
    install_sound(DIGI_AUTODETECT, MIDI_NONE, NULL);
        stream = play_audio_stream(11025, 8, FALSE, 22050, 255, 128);


       p=(unsigned char*)get_audio_stream_buffer(stream);
    valor=0x00;
   if (p) {
                for(offset=0;offset<11025;offset++){
            p[offset] = valor;
            valor+=incremento;
                }
        }
                 free_audio_stream_buffer(stream);

   timer=time(NULL);
        offset=0;


 return 1;
}
```

Fichero clicksample.c

```
#ifdef _WIN32
#  include <windows.h>
#endif
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>

#ifndef __APPLE__
#  include <GL/glut.h>
#else
#  include <GLUT/glut.h>
#endif
#include <AR/gsub.h>
#include <AR/param.h>
#include <AR/ar.h>
#include <AR/video.h>
#include <time.h>
#include <formatopixel.h>




#define MIN_H 0
#define MAX_H 0
#define MIN_S 0
#define MAX_S 0
#define MIN_B 0
#define MAX_B 0
#define UMBRAL 5


int             xsize, ysize;
int                                     thresh = 100;
int             count = 0;

//centro y radio del circulo
double cx=0.5;
double cy=0.5;
double r=0.02;
int punto[2];
int click;
int arrastrar=0;
long timer;

int             minH = MIN_H, maxH = MAX_H, minS = MIN_S, maxS = MAX_S, minB = MIN_B,
maxB = MAX_B, umbral = UMBRAL;
int invert = 0;
ARUint8         *dataPtr;
```

```
/* set video capture configuration */
#ifdef _WIN32
char                    *vconf = "flipV,flipH,showDlg"; // see video.h for a list of supported
parameters
#else
char                    vconf[256];
#endif

char          *cparam_name    = "Data/camera_para.dat";
ARParam         cparam;


static void   init(void);
static void   cleanup(void);
static void   keyEvent( unsigned char key, int x, int y);
static void   mouseEvent(int button, int state, int x, int y);
static void   mainLoop(void);
static int    guardarEnFichero(void);
static int    cargarDeFichero(void);
static void   dibujaPelota(void);
//static void   dibujaObjetivos(void);

int main(int argc, char** args)
{
        //initialize applications

#ifndef _WIN32
char dev[256];
strcpy(vconf,"-dev=camara -channel=0 -width=320 -height=240 -palette=YUV420P");


if(argc==2) {
 strcpy(vconf,"-channel=0 -palette=YUV420P -width=320 -height=240 -dev=");
 strcat(vconf, args[1]);
}
else {
 FILE *f;
 f=fopen("camara","r");
 if(f == NULL) {
  printf("No se ha encontrado un enlace \"./camara\" al dispositivo de video. Por favor, indique el
nombre de dispositivo (normalmente /dev/video0 o /dev/video1) en el que se encuentra su camara:
");
  scanf("%s",&dev);
  strcpy(vconf, "-channel=0 -palette=YUV420P -width=320 -height=240 -dev=");
  strcat(vconf, dev);


 }
 else {
 fclose(f);
 }

}
```

```
#endif

    init();

        //start video capture
        arVideoCapStart();

        //start the main event loop
    argMainLoop( mouseEvent, keyEvent, mainLoop );

        return 0;
}

static void   keyEvent( unsigned char key, int x, int y)
{    char resp;
    /* quit if the ESC key is pressed */
    if( key == 0x1b ) {
       cleanup();
       exit(0);
    }

        if( key == 'h' ) {
                printf("\nminH = %d, maxH = %d\n",minH,maxH);
           printf("\nIntroduce el valor minimo de H(tono):\n");
                scanf("%d",&minH);
           printf("\nIntroduce el valor maximo de H(tono):\n");
                scanf("%d",&maxH);

        }

    if( key == 'b' ) {
           printf("\nminB = %d, maxB = %d\n",minB,maxB);
           printf("\nIntroduce el valor minimo de B(brillo):\n");
                scanf("%d",&minB);
           printf("\nIntroduce el valor maximo de B(brillo):\n");
                scanf("%d",&maxB);
        }

    if( key == 's' ) {
                printf("\nminS = %d, maxS = %d\n",minS,maxS);
           printf("\nIntroduce el valor minimo de S(Saturacion):\n");
                scanf("%d",&minS);
           printf("\nIntroduce el valor maximo de S(Saturacion):\n");
                scanf("%d",&maxS);

        }

         if( key == 'i' ) {

                 if(invert) invert = 0;
                  else invert = 1;
            printf("\nInversion realizada\n");
```

```
        }

        if( key == 'u' ) {

           printf("\nIntroduce el umbral:\n");
                scanf("%d",&umbral);

        }

    if( key == 'g' ) {

           printf("\nValores actuales: \nH: [%d-%d]\nS: [%d-%d]\nB: [%d-%d]
\n",minH,maxH,minS,maxS,minB,maxB);
           printf("\nDesea guardar los datos de configuracion de la sesion actual? (s/n)\n");
                scanf("%c",&resp);
                if(resp=='s') {
       if(guardarEnFichero()) printf("\nConfiguracion guardada en fichero.\n");
                 else {printf("\nError!. No se pudo guardar la configuracion en el fichero.\n");}
                }


        }

}

static void mouseEvent(int button, int state, int x, int y) {
  double H, S, V;
  ARUint8 *pnt;
  x=x/2;
  y=y/2;

  //varios clicks para autoajustar los valores H, S y V

if (button == GLUT_LEFT_BUTTON && state == GLUT_DOWN) {
pnt = (dataPtr+(arImXsize*VALORES_POR_PIXEL*(y-1))+(x*VALORES_POR_PIXEL));
                RGBtoHSV(R_COMP, G_COMP, B_COMP, &H, &S, &V);

                if(H>=(maxH-umbral)) {maxH=(H+umbral);}
                if(H<=(minH+umbral)) {minH=(H-umbral);}
              if(S>=(maxS-umbral)) {maxS=(S+umbral);}
                if(S<=(minS+umbral)) {minS=(S+umbral);}
                if(V>=(maxB-umbral)) {maxB=(V+umbral);}
                if(V<=(minB+umbral)) {minB=(V-umbral);}

                printf("\nValores del pixel:\n%f %f %f\n",H, S, V);
                printf("\nNuevos rangos: \nH: [%d-%d]\nS: [%d-%d]\nB: [%d-%d]
\n",minH,maxH,minS,maxS,minB,maxB);

    }

  //para declarar el punto inicial del objetivo
  if (button == GLUT_RIGHT_BUTTON && state == GLUT_DOWN) {
pnt = (dataPtr+(arImXsize*VALORES_POR_PIXEL*(y-1))+(x*VALORES_POR_PIXEL));
```

```
        RGBtoHSV(R_COMP, G_COMP, B_COMP, &H, &S, &V);


            minH=H-umbral;
                maxH=H+umbral;
                minS=S-umbral;
                maxS=S+umbral;
                minB=S-umbral;
                maxB=S+umbral;
                printf("\nValores del pixel:\n%f %f %f\n",H, S, V);


    }

}




/* main loop */
static void mainLoop(void)
{

//    ARMarkerInfo    *marker_info;
          ARMarkerInfo2   *marker2;
          int             marker_num2;
    //int           marker_num;
    int           i,j;
          ARInt16         *limage;
    int           label_num;
    int           *area, *clip, *label_ref;
    double          *pos;



    marker_num2 = 0;


    /* grab a video frame */
    if( (dataPtr = (ARUint8 *)arVideoGetImage()) == NULL ) {
        arUtilSleep(2);
        return;
    }

    if( count == 0 ) arUtilTimerReset();
    count++;

          /*draw the video*/
    argDrawMode2D();
    argDispImage( dataPtr, 0,0 );
```

```
        /* capture the next video frame */
        arVideoCapNext();

        glColor3f( 1.0, 0.0, 0.0 );
        glLineWidth(6.0);

        /* detect the markers in the video frame */
//      if(arDetectMarker(dataPtr, thresh,
//                &marker_info, &marker_num, &marker2) < 0 ) {

//          cleanup();
//                exit(0);
//      }

   limage = arLabelingHSB( dataPtr, thresh,
                &label_num, &area, &pos, &clip, &label_ref, 1, minH, maxH, minS, maxS, minB,
maxB, invert);
   if( limage == 0 )   {
          cleanup();
          exit(0);

        }

   marker2 = arDetectMarker3( limage, label_num, label_ref,
                        area, pos, clip, AR_AREA_MAX, AR_AREA_MIN, &marker_num2);
   if( marker2 == 0 ) {
          cleanup();
          exit(0);

        }


        if(timer==0 && marker_num2==0) {
         click=0;
         timer = (long) time(NULL);
        }
        else if (timer>0 && marker_num2>0){
                if((time(NULL)-timer)<1){click=1;}
         timer = 0;
        }
        else {
         click=0;
        }

        for( i = 0; i < marker_num2; i++ ) {
                for(j=0; j<marker2[i].coord_num-1; j++) {
                 argLineSeg( marker2[i].x_coord[j] ,  marker2[i].y_coord[j],  marker2[i].x_coord
[j+1],  marker2[i].y_coord[j+1], 0, 0);


                 if(click) {
                  if(abs((int)(marker2[i].x_coord[j]-(cx*arImXsize)))<(r*arImXsize) && abs((int)
(marker2[i].y_coord[j]-(arImYsize-(cy*arImYsize))))<(r*arImYsize)){
```

```
                        arrastrar=!arrastrar;
                        punto[0]=marker2[i].x_coord[j];
                        punto[1]=marker2[i].y_coord[j];
                        break;
                    }//end_if
                    else {
                    //arrastrar=0; //debe soltarse cuando se haga click fuera del objeto?, pero si esta arrastrandolo siempre estara dentro si no se sale de la pantalla.
                    }
                   }//end_if
                  }//end_for
          }//end_for


   if(arrastrar) { cx=(double)marker2[0].pos[0]/(double)arImXsize; cy=1.0-((double)marker2[0].pos[1]/(double)arImYsize);}

          dibujaPelota();
          //dibujaObjetivos();
          argSwapBuffers();
}

static void init( void )
{
          ARParam  wparam;

           click=0;
           timer=time(NULL);


          cargarDeFichero();
          /* selecciona el color de borrado */
   glClearColor (0.0, 0.0, 0.0, 0.0);
   /* open the video path */
   if( arVideoOpen( vconf ) < 0 ) exit(0);
   /* find the size of the window */
   if( arVideoInqSize(&xsize, &ysize) < 0 ) exit(0);
   printf("Image size (x,y) = (%d,%d)\n", xsize, ysize);

   /* set the initial camera parameters */
   if( arParamLoad(cparam_name, 1, &wparam) < 0 ) {
      printf("Camera parameter load error !!\n");
      exit(0);
   }
   arParamChangeSize( &wparam, xsize, ysize, &cparam );
   arInitCparam( &cparam );
   printf("*** Camera Parameter ***\n");
   arParamDisp( &cparam );



   /* open the graphics window */
   argInit( &cparam, 2.0, 0, 0, 0, 0 );
```

```
}

/* cleanup function called when program exits */
static void cleanup(void)
{
        arVideoCapStop();
   arVideoClose();
   argCleanup();
}

/* draw the the AR objects */



static int cargarDeFichero(void) {
 FILE *fich;
 char c;
 fich=fopen("conf.con","r");

 if(fich != NULL) {

        fscanf(fich,"%d",&umbral);
        fscanf(fich," %c",&c);
    printf("%c",c);

        if(c=='#'){  //si el separador es correcto y no es fin de fichero carga valores;
         fscanf(fich,"%d",&minH);
         fscanf(fich,"%d",&maxH);
         fscanf(fich,"%d",&minS);
         fscanf(fich,"%d",&maxS);
         fscanf(fich,"%d",&minB);
         fscanf(fich,"%d",&maxB);
        }
  fclose(fich);
  return 1;
 }
 return 0;

}

static int guardarEnFichero(void) {
 FILE *fich;

 fich=fopen("conf.con","w");

 if(fich != NULL) {
  fprintf(fich,"%d\n#\n%d %d\n%d %d\n%d %d\n-
\n",umbral,minH,maxH,minS,maxS,minB,maxB);
  fclose(fich);
  return 1;
 }
 return 0;
```

```
}

static void dibujaPelota() {


int i;

/* inicializa los valores de la vista */
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluOrtho2D(0.0, 1.0, 0.0, 1.0);

glColor3f(0, 0, 1.0);

glBegin(GL_POLYGON);

for (i = 0; i < 100 + 1; i++) { // +1 para cerrar
 glVertex2f( cx + r * cos(2.0 * 3.14159 * i / 100), cy + r * sin(2.0 * 3.14159 * i / 100) );
 }
glEnd();

/* Vacia el buffer de dibujo */
glFlush ();

}
```

Fichero sample.c

```
#ifdef _WIN32
#  include <windows.h>
#endif
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>

#ifndef __APPLE__
#  include <GL/glut.h>
#else
#  include <GLUT/glut.h>
#endif
#include <AR/gsub.h>
#include <AR/param.h>
#include <AR/ar.h>
#include <AR/video.h>
#include <time.h>
#include <formatopixel.h>



#define MIN_H 0
#define MAX_H 0
#define MIN_S 0
#define MAX_S 0
#define MIN_B 0
#define MAX_B 0

#define MAX_VEL 3*r
#define MAX_ACC 0.007
#define UMBRAL 5


int		xsize, ysize;
int                             thresh = 100;
int		count = 0;

//centro y radio del circulo
double cx=0.5;
double cy=0.5;
double velx;
double vely;
int dirx;
int diry;
double accx, accy;
double r=0.02;
int dentro = 0;
int punto[2];
```

```
int           minH = MIN_H, maxH = MAX_H, minS = MIN_S, maxS = MAX_S, minB = MIN_B,
maxB = MAX_B, umbral = UMBRAL;
int invert = 0;
ARUint8         *dataPtr;


/* set video capture configuration */
#ifdef _WIN32
char                    *vconf = "flipV,flipH,showDlg"; // see video.h for a list of supported
parameters
#else
char                    vconf[256];
#endif

char          *cparam_name    = "Data/camera_para.dat";
ARParam         cparam;


static void   init(void);
static void   cleanup(void);
static void   keyEvent( unsigned char key, int x, int y);
static void   mouseEvent(int button, int state, int x, int y);
static void   mainLoop(void);
static int    guardarEnFichero(void);
static int    cargarDeFichero(void);
static void   dibujaPelota(void);


int main(int argc, char** args)
{
        //initialize applications

#ifndef _WIN32
char dev[256];
strcpy(vconf,"-dev=camara -channel=0 -width=320 -height=240 -palette=YUV420P");


if(argc==2) {
 strcpy(vconf,"-channel=0 -palette=YUV420P -width=320 -height=240 -dev=");
 strcat(vconf, args[1]);
}
else {
 FILE *f;
 f=fopen("camara","r");
 if(f == NULL) {
  printf("No se ha encontrado un enlace \"./camara\" al dispositivo de video. Por favor, indique el
nombre de dispositivo (normalmente /dev/video0 o /dev/video1) en el que se encuentra su camara:
");
  scanf("%s",&dev);
  strcpy(vconf, "-channel=0 -palette=YUV420P -width=320 -height=240 -dev=");
  strcat(vconf, dev);
```

```
 }
 else {
 fclose(f);
 }

}
#endif


   init();

         //start video capture
         arVideoCapStart();

         //start the main event loop
   argMainLoop( mouseEvent, keyEvent, mainLoop );

         return 0;
}

static void   keyEvent( unsigned char key, int x, int y)
{    char resp;
   /* quit if the ESC key is pressed */
   if( key == 0x1b ) {
      cleanup();
      exit(0);
   }

         if( key == 'h' ) {
                  printf("\nminH = %d, maxH = %d\n",minH,maxH);
            printf("\nIntroduce el valor minimo de H(tono):\n");
                  scanf("%d",&minH);
            printf("\nIntroduce el valor maximo de H(tono):\n");
                  scanf("%d",&maxH);

         }

   if( key == 'b' ) {
            printf("\nminB = %d, maxB = %d\n",minB,maxB);
            printf("\nIntroduce el valor minimo de B(brillo):\n");
                  scanf("%d",&minB);
            printf("\nIntroduce el valor maximo de B(brillo):\n");
                  scanf("%d",&maxB);
         }

   if( key == 's' ) {
                  printf("\nminS = %d, maxS = %d\n",minS,maxS);
            printf("\nIntroduce el valor minimo de S(Saturacion):\n");
                  scanf("%d",&minS);
            printf("\nIntroduce el valor maximo de S(Saturacion):\n");
                  scanf("%d",&maxS);

         }
```

```
   if( key == 'r' ) {
          cx=0.5;
          cy=0.5;
          srand( (unsigned int) time( NULL ) );
    velx=0.01;
          vely=0.02;
          accx=0;
          accy=0;
          dirx=1;diry=1;
          if(rand()%2<1) dirx=-1;
          if(rand()%2<1) diry=-1;

          }



          if( key == 'i' ) {

                   if(invert) invert = 0;
                    else invert = 1;
              printf("\nInversion realizada\n");

          }

          if( key == 'u' ) {

             printf("\nIntroduce el umbral:\n");
                   scanf("%d",&umbral);

          }

   if( key == 'g' ) {

             printf("\nValores actuales: \nH: [%d-%d]\nS: [%d-%d]\nB: [%d-%d]
\n",minH,maxH,minS,maxS,minB,maxB);
             printf("\nDesea guardar los datos de configuracion de la sesion actual? (s/n)\n");
                   scanf("%c",&resp);
                   if(resp=='s') {
       if(guardarEnFichero()) printf("\nConfiguracion guardada en fichero.\n");
                    else {printf("\nError!. No se pudo guardar la configuracion en el fichero.\n");}
                   }


          }

}
static void mouseEvent(int button, int state, int x, int y) {
   double H, S, V;
  ARUint8 *pnt;
  x=x/2;
  y=y/2;
```

```
  //varios clicks para autoajustar los valores H, S y V

if (button == GLUT_LEFT_BUTTON && state == GLUT_DOWN) {
pnt = (dataPtr+(arImXsize*VALORES_POR_PIXEL*(y-1))+(x*VALORES_POR_PIXEL));
                RGBtoHSV(R_COMP, G_COMP, B_COMP, &H, &S, &V);

                if(H>=(maxH-umbral)) {maxH=(H+umbral);}
                if(H<=(minH+umbral)) {minH=(H-umbral);}
              if(S>=(maxS-umbral)) {maxS=(S+umbral);}
                if(S<=(minS+umbral)) {minS=(S+umbral);}
                if(V>=(maxB-umbral)) {maxB=(V+umbral);}
                if(V<=(minB+umbral)) {minB=(V-umbral);}

                printf("\nValores del pixel:\n%f %f %f\n",H, S, V);
                printf("\nNuevos rangos: \nH: [%d-%d]\nS: [%d-%d]\nB: [%d-%d]
\n",minH,maxH,minS,maxS,minB,maxB);

   }

  //para declarar el punto inicial del objetivo
  if (button == GLUT_RIGHT_BUTTON && state == GLUT_DOWN) {
pnt = (dataPtr+(arImXsize*VALORES_POR_PIXEL*(y-1))+(x*VALORES_POR_PIXEL));

        RGBtoHSV(R_COMP, G_COMP, B_COMP, &H, &S, &V);


           minH=H-umbral;
                maxH=H+umbral;
                minS=S-umbral;
                maxS=S+umbral;
                minB=S-umbral;
                maxB=S+umbral;
                printf("\nValores del pixel:\n%f %f %f\n",H, S, V);


   }


}


/* main loop */
static void mainLoop(void)
{

//   ARMarkerInfo    *marker_info;
        ARMarkerInfo2   *marker2;
        int             marker_num2;
   //int           marker_num;
   int            i,j;
        ARInt16         *limage;
   int            label_num;
```

```
    int             *area, *clip, *label_ref;
    double          *pos;



    marker_num2 = 0;


    /* grab a video frame */
    if( (dataPtr = (ARUint8 *)arVideoGetImage()) == NULL ) {
       arUtilSleep(2);
       return;
    }

    if( count == 0 ) arUtilTimerReset();
    count++;

          /*draw the video*/
    argDrawMode2D();
    argDispImage( dataPtr, 0,0 );

          /* capture the next video frame */
          arVideoCapNext();

          glColor3f( 1.0, 0.0, 0.0 );
          glLineWidth(6.0);

          /* detect the markers in the video frame */
//        if(arDetectMarker(dataPtr, thresh,
//                  &marker_info, &marker_num, &marker2) < 0 ) {

//           cleanup();
//                  exit(0);
//        }

    limage = arLabelingHSB( dataPtr, thresh,
                   &label_num, &area, &pos, &clip, &label_ref, 1, minH, maxH, minS, maxS, minB,
maxB, invert);
    if( limage == 0 )   {
             cleanup();
             exit(0);

          }

    marker2 = arDetectMarker3( limage, label_num, label_ref,
                              area, pos, clip, AR_AREA_MAX, AR_AREA_MIN, &marker_num2);
    if( marker2 == 0 ) {
             cleanup();
             exit(0);

          }
          dentro=0;
          for( i = 0; i < marker_num2; i++ ) {
```

```
                for(j=0; j<marker2[i].coord_num-1; j++) {
                 argLineSeg( marker2[i].x_coord[j] ,  marker2[i].y_coord[j],  marker2[i].x_coord
[j+1],  marker2[i].y_coord[j+1], 0, 0);

                 if(abs((int)(marker2[i].x_coord[j]-(cx*arImXsize)))<(r*arImXsize) && abs((int)
(marker2[i].y_coord[j]-(arImYsize-(cy*arImYsize))))<(r*arImYsize)){
                        dentro=1;
                        punto[0]=marker2[i].x_coord[j];
                        punto[1]=marker2[i].y_coord[j];

                 }
                }
                if(dentro) {
                        if(punto[0]>cx*arImXsize) {dirx=-1;}
                        else {dirx=1;}
                         if(punto[1]>(arImYsize-cy*arImYsize)) {dirx=-1;}
                         else {diry=1;}

                         if(accx>0 && accy>0) {accx=accx*1.01; accy=accy*1.01;}
                         else if(accx<=MAX_ACC && accy<=MAX_ACC) {accy=0.001;
accy=0.001;}

                }//end_if
        }//end_for

        /*swap the graphics buffers*/
        if(velx<=MAX_VEL){
         velx+=accx;
        }
        if(vely<=MAX_VEL){
         vely+=accy;
        }

        cx+=velx*dirx;
        cy+=vely*diry;
        if((cx*arImXsize+(arImXsize*r))>=arImXsize && dirx==1) {dirx=-1;}
        if((cx*arImXsize-(arImXsize*r))<=0 && dirx==-1) {dirx=1;}
        if((cy*arImYsize+(arImYsize*r))>=arImYsize && diry==1) diry=-1;
        else if((cy*arImYsize-(arImYsize*r))<=0) {cx=0.5;cy=0.5; accx=0; accy=0; velx=0.01,
vely=0.02;}
        dibujaPelota();
        argSwapBuffers();
}

static void init( void )
{
        ARParam  wparam;
         srand( (unsigned int) time( NULL ) );
    velx=0.01;
         vely=0.02;
         accx=0;
         accy=0;
         dirx=1;diry=1;
```

```
            if(rand()%2<1) dirx=-1;
            if(rand()%2<1) diry=-1;



           cargarDeFichero();
           /* selecciona el color de borrado */
    glClearColor (0.0, 0.0, 0.0, 0.0);
    /* open the video path */
    if( arVideoOpen( vconf ) < 0 ) exit(0);
    /* find the size of the window */
    if( arVideoInqSize(&xsize, &ysize) < 0 ) exit(0);
    printf("Image size (x,y) = (%d,%d)\n", xsize, ysize);

    /* set the initial camera parameters */
    if( arParamLoad(cparam_name, 1, &wparam) < 0 ) {
       printf("Camera parameter load error !!\n");
       exit(0);
    }
    arParamChangeSize( &wparam, xsize, ysize, &cparam );
    arInitCparam( &cparam );
    printf("*** Camera Parameter ***\n");
    arParamDisp( &cparam );



    /* open the graphics window */
    argInit( &cparam, 2.0, 0, 0, 0, 0 );
}

/* cleanup function called when program exits */
static void cleanup(void)
{
           arVideoCapStop();
    arVideoClose();
    argCleanup();
}

/* draw the the AR objects */



static int cargarDeFichero(void) {
 FILE *fich;
 char c;
 fich=fopen("conf.con","r");

 if(fich != NULL) {

            fscanf(fich,"%d",&umbral);
            fscanf(fich," %c",&c);
     printf("%c",c);
```

```
            if(c=='#'){  //si el separador es correcto y no es fin de fichero carga valores;
             fscanf(fich,"%d",&minH);
             fscanf(fich,"%d",&maxH);
             fscanf(fich,"%d",&minS);
             fscanf(fich,"%d",&maxS);
             fscanf(fich,"%d",&minB);
             fscanf(fich,"%d",&maxB);
            }
  fclose(fich);
  return 1;
 }
 return 0;

}

static int guardarEnFichero(void) {
 FILE *fich;

 fich=fopen("conf.con","w");

 if(fich != NULL) {
  fprintf(fich,"%d\n#\n%d %d\n%d %d\n%d %d\n-
\n",umbral,minH,maxH,minS,maxS,minB,maxB);
  fclose(fich);
  return 1;
 }
 return 0;

}

static void dibujaPelota() {


int i;

/* inicializa los valores de la vista */
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluOrtho2D(0.0, 1.0, 0.0, 1.0);

glColor3f(0, 0, 1.0);

glBegin(GL_POLYGON);

for (i = 0; i < 100 + 1; i++) { // +1 para cerrar
 glVertex2f( cx + r * cos(2.0 * 3.14159 * i / 100), cy + r * sin(2.0 * 3.14159 * i / 100) );
 }
glEnd();

/* Vacia el buffer de dibujo */
glFlush ();

}
```

Fichero pelotita.c

```
#ifdef _WIN32
#  include <windows.h>
#endif
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>

#ifndef __APPLE__
#  include <GL/glut.h>
#else
#  include <GLUT/glut.h>
#endif
#include <AR/gsub.h>
#include <AR/param.h>
#include <AR/ar.h>
#include <AR/video.h>
#include <time.h>
#include <formatopixel.h>

#define PLAYER_HEIGHT  0.04
#define PLAYER_WIDTH  0.12
#define MIN_H 0
#define MAX_H 0
#define MIN_S 0
#define MAX_S 0
#define MIN_B 0
#define MAX_B 0

#define MAX_VEL 2*r
#define UMBRAL 5


int           xsize, ysize;
int                                   thresh = 100;
int           count = 0;

//centro y radio del circulo
double cx=0.5;
double cy=0.5;
double velx;
double vely;
int dirx;
int diry;
int puntos;
int antPuntos=0;
double accx, accy;
double r=0.02;
int dentro = 0;
int punto[2];
```

```
int           minH = MIN_H, maxH = MAX_H, minS = MIN_S, maxS = MAX_S, minB = MIN_B,
maxB = MAX_B, umbral = UMBRAL;
int invert = 0;
ARUint8         *dataPtr;


/* set video capture configuration */
#ifdef _WIN32
char                    *vconf = "flipV,flipH,showDlg"; // see video.h for a list of supported
parameters
#else
char                    vconf[256];
#endif

char          *cparam_name    = "Data/camera_para.dat";
ARParam         cparam;


typedef struct {
   GLubyte  *dibujo;       // Un puntero a los datos de la imagen
        GLuint          bpp;            // bpp significa bits per pixel (bits por punto) es la calidad en
palabras sensillas
        GLuint largo;           // Largo de la textura
        GLuint ancho;                   // Ancho de la textura
        GLuint ID;              // ID de la textura, es como su nombre para opengl
}textura;


textura texturas[2];

static void   init(void);
static void   cleanup(void);
static void   keyEvent( unsigned char key, int x, int y);
static void   mainLoop(void);
static int    guardarEnFichero(void);
static int    cargarDeFichero(void);
static void   dibujaPelota();
static void   dibujaJugador(double x, double y);
static int    colisionJugador(double x, double y, double cx, double cy, double r);
int cargarTGA( char *nombre, textura *imagen) ;
void cambiarTamano(int largo, int ancho);
//static void   dibujaObjetivos(void);



int cargarTGA( char *nombre, textura *imagen) {
        GLubyte                 cabezeraTGA[12]={0,0,2,0,0,0,0,0,0,0,0,0};                  //
Cabezera de un tga sin compresion
      GLubyte       compararTGA[12];                                // Aca vamos a comprar la
cabezera
        GLubyte                 cabezera[6];                                            // Los 6 bytes
importantes
        GLuint                  bytesporpunto;                                          // Cuantos
```

```
bytes hay por punto
        GLuint          tamanoimagen;                                   // Aca
guardaremos el tamaï¿½ de la imagen
        GLuint          temp,i;                                         // Variable
temporal, y una para usar con el for
        GLuint          tipo=GL_RGBA;                                   // Nuestro
typo por defecto, lo veremos mas abajo

        FILE *archivo=fopen(nombre, "rb");          // Cargamos nuestro archivo en memoria
        if( archivo == NULL ||                      // Existe nuestro archivo??
        fread(compararTGA,1,sizeof(compararTGA),archivo)!=sizeof(compararTGA)  ||     // Hay
12 bytes para leer??
           memcmp(cabezeraTGA,compararTGA,sizeof(compararTGA))!=0            ||        //
Son iguales??
           fread(cabezera,1,sizeof(cabezera),archivo)!=sizeof(cabezera)) {
                if(archivo==NULL) {
                        printf("No se encontro el archivo %s\n",nombre);
                        return 0;               // No se abrio el archivo
                }
                else {
                        fclose(archivo);
                        return 0;
                }
        }
        /* Ahora hay que leer la cabezera del archivo, para saber cuanto es el largo, ancho, y los
bytes por puntos,
        para eso aca hay una ilustracion de la cabezera :
        6 bytes -> xxxxxxx xxxxxxx xxxxxxx xxxxxxx xxxxxxx xxxxxxx
                   |--- Largo ---| |---Ancho-----| |-bpp-|
        El dato del largo se guarda en el cabezera[0] y cabezera[1], para leerlo hay que multiplicar
cabezera[0] por 256 y           sumarselo a cabezera[1], para leer ancho y bpp es el mismo
procedimiento */
        imagen->largo=256*cabezera[1]+ cabezera[0];
        imagen->ancho=256*cabezera[3]+ cabezera[2];

        /* Ahora vemos si hay datos no validos, como largo o ancho iguales menores a 0 o iguales
a 0 */
        if( imagen->largo <= 0 ||       // Largo mayor que 0??
           imagen->ancho <= 0 ||       // Ancho mayor que 0??
           (cabezera[4]!=24 && cabezera[4]!=32)) {          // bpp es 24 o 32?? (solo se cargan 24 y
32 bpp)
           printf("Daton invalidos\n");
                fclose(archivo);
                return 0;
        }
        imagen->bpp=cabezera[4];    // Aca se guardan los bits por punto
        bytesporpunto=cabezera[4]/8;          // Aca los bytes por punto (1 byte = 8 bits)
      tamanoimagen=imagen->largo * imagen->ancho * bytesporpunto;      // Esta es la memoria
que nesecitaremos para guardar los datos de la textura
        /*Ahora reservamos espacio en memoria para nuestra textura, luego leemos la textura del
archivo */
        imagen->dibujo = (GLubyte *)malloc(tamanoimagen);       // Reservamos la memoria
necesaria para nuestra textura
```

```
        if(imagen->dibujo== NULL ||                     // Se logro reservar la memoria???
          fread(imagen->dibujo, 1, tamanoimagen, archivo) != tamanoimagen ) { // Se lee, y se
comprueba que lo leido es de la misma longitud que la asignada a a dibujo.
                if(imagen->dibujo != NULL) {
                        printf("Error leyendo imagen\n");
                        free(imagen->dibujo);
                } else printf("Error asignando memoria\n");
                fclose(archivo);
                return 0;
        }
        /* El formato tga guarda las imagenes en BGR, y opengl usa RGB,por lo cambiamos de
lugares */
      for(i=0; i< (int)tamanoimagen; i+=bytesporpunto)
        {
                temp=imagen->dibujo[i];                                 // Guardamos el primer valor
                imagen->dibujo[i] = imagen->dibujo[i + 2];          // Asignamos el nuevo primer
valor
                imagen->dibujo[i + 2] = temp;                           // Asignamos el ultimo valor
        }

        fclose (archivo);       // Cerramos el archivo

        /* Listo, terminamos con el codigo de carga, volvemos a opengl, ahora hay que asignarle a
la textura un ID, luego         decirle a opengl cuales son el largo, el ancho y los bpp */

      glGenTextures( 1 , &imagen[0].ID);    // Crea un ID para la textura, buscando un id que este
vacio
      glBindTexture(GL_TEXTURE_2D, imagen[0].ID);       // Seleccionamos nuestra textura
        if(imagen->bpp ==24) tipo= GL_RGB;      // Si nuestra textura es de 24 bits, entonces se
crea una textura rgb, sino una rgba
        /* Ahora creamos nuestra textura, entrando el largo, ancho y tipo */
      glTexImage2D(GL_TEXTURE_2D, 0, tipo, imagen[0].ancho, imagen[0].largo, 0, tipo,
GL_UNSIGNED_BYTE, imagen[0].dibujo);
        /* Ahora le decimos a opengl como queremos que se vea nuestra textura, MAG_FILTER es
cuando la textura es mas grande que el lugar donde la asignamos, y MIG_FILTER, es cuando la
textura es mas pequeï¿½ que el lugar donde la asignamos, GL_LINEAR es para que se vea bien
tanto cerca como lejos, pero ocupa bastante procesador. Otra opcion el GL_NEARES, que ocupa
menos prosesador */
        glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
        glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
        return 1;       // Todo salio bien
}


void cambiarTamano(int largo, int ancho) {
      if(ancho==0) ancho=1;         // Previene que dividamos por 0
      glMatrixMode(GL_PROJECTION);  // Escojemos la matriz de proyeccion
      glLoadIdentity();             // Se resetea la matriz
      glViewport(0,0,largo, ancho);   // Se va a usar toda la ventana para mostrar graficos
      gluPerspective( 45 ,          // Angulo de vision
                (float)largo/(float)ancho,
                1.0,            // Cuan cerca se puede ver
                1000);          // Cuan lejos se puede ver
```

```
	glMatrixMode(GL_MODELVIEW);     // Escojemos la matriz de vista
	   glLoadIdentity();                  // Se resetea la matriz
	gluLookAt( 0.0, 0.0, 0.0,       // Hacia donde miramos
	        0.0, 0.0, -3.0,      // Desde donde miramos
	        0.0, 1.0, 0.0);      // Que eje es el que esta hacia arriba
}


int main(int argc, char** args)
{
	//initialize applications
#ifndef _WIN32
char dev[256];
strcpy(vconf,"-dev=camara -channel=0 -width=320 -height=240 -palette=YUV420P");


if(argc==2) {
 strcpy(vconf,"-channel=0 -palette=YUV420P -width=320 -height=240 -dev=");
 strcat(vconf, args[1]);
}
else {
 FILE *f;
 f=fopen("camara","r");
 if(f == NULL) {
  printf("No se ha encontrado un enlace \"./camara\" al dispositivo de video. Por favor, indique el
nombre de dispositivo (normalmente /dev/video0 o /dev/video1) en el que se encuentra su camara:
");
  scanf("%s",&dev);
  strcpy(vconf, "-channel=0 -palette=YUV420P -width=320 -height=240 -dev=");
  strcat(vconf, dev);


 }
 else {
 fclose(f);
 }

}
#endif

   init();

	//start video capture
	arVideoCapStart();

	//start the main event loop
   argMainLoop( NULL, keyEvent, mainLoop );

	return 0;
}

static void keyEvent( unsigned char key, int x, int y)
{    char resp;
```

```
/* quit if the ESC key is pressed */
if( key == 0x1b ) {
   cleanup();
   exit(0);
}


      if( key == 'h' ) {
            printf("\nminH = %d, maxH = %d\n",minH,maxH);
         printf("\nIntroduce el valor minimo de H(tono):\n");
            scanf("%d",&minH);
         printf("\nIntroduce el valor maximo de H(tono):\n");
            scanf("%d",&maxH);

      }

if( key == 'b' ) {
         printf("\nminB = %d, maxB = %d\n",minB,maxB);
         printf("\nIntroduce el valor minimo de B(brillo):\n");
            scanf("%d",&minB);
         printf("\nIntroduce el valor maximo de B(brillo):\n");
            scanf("%d",&maxB);
      }

if( key == 's' ) {
            printf("\nminS = %d, maxS = %d\n",minS,maxS);
         printf("\nIntroduce el valor minimo de S(Saturacion):\n");
            scanf("%d",&minS);
         printf("\nIntroduce el valor maximo de S(Saturacion):\n");
            scanf("%d",&maxS);

      }


if( key == 'r' ) {


       cx=0.5;
       cy=0.5;
       srand( (unsigned int) time( NULL ) );
 velx=0.01;
       vely=0.03;
       accx=0;
       accy=0;
       dirx=1;diry=1;
       if(rand()%2<1) dirx=-1;
       if(rand()%2<1) diry=-1;

      }



       if( key == 'i' ) {
```

```
                    if(invert) invert = 0;
                     else invert = 1;
                printf("\nInversion realizada\n");

            }

            if( key == 'u' ) {

               printf("\nIntroduce el umbral:\n");
                    scanf("%d",&umbral);

            }

    if( key == 'g' ) {

               printf("\nValores actuales: \nH: [%d-%d]\nS: [%d-%d]\nB: [%d-%d]
\n",minH,maxH,minS,maxS,minB,maxB);
               printf("\nDesea guardar los datos de configuracion de la sesion actual? (s/n)\n");
                    scanf("%c",&resp);
                    if(resp=='s') {
          if(guardarEnFichero()) printf("\nConfiguracion guardada en fichero.\n");
                     else {printf("\nError!. No se pudo guardar la configuracion en el fichero.\n");}
                    }


            }

}

/* main loop */
static void mainLoop(void)
{

//    ARMarkerInfo    *marker_info;
          ARMarkerInfo2   *marker2;
          int             marker_num2;
    //int           marker_num;
    int           i;
          ARInt16         *limage;
    int           label_num;
    int           *area, *clip, *label_ref;
    double          *pos;
    int                             direccion;


    marker_num2 = 0;


    /* grab a video frame */
    if( (dataPtr = (ARUint8 *)arVideoGetImage()) == NULL ) {
       arUtilSleep(2);
       return;
```

```
    }

    if( count == 0 ) arUtilTimerReset();
    count++;

        glClear (GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    /* capture the next video frame */
        arVideoCapNext();

        glColor3f( 1.0, 0.0, 0.0 );
        glLineWidth(6.0);


    limage = arLabelingHSB( dataPtr, thresh,
                    &label_num, &area, &pos, &clip, &label_ref, 1, minH, maxH, minS, maxS, minB,
maxB, invert);
    if( limage == 0 )   {
            cleanup();
            exit(0);

        }

    marker2 = arDetectMarker3( limage, label_num, label_ref,
                            area, pos, clip, AR_AREA_MAX, AR_AREA_MIN, &marker_num2);
    if( marker2 == 0 ) {
            cleanup();
            exit(0);

        }
        dentro=0;
        for( i = 0; i < marker_num2; i++ ) {

                            dibujaJugador(marker2[i].pos[0]/arImXsize,1-marker2[i].pos[1]/arImYsize);
        direccion=colisionJugador(marker2[i].pos[0]/arImXsize,1-marker2[i].pos[1]/
arImYsize,cx,cy,r);

                            switch(direccion) {

                            case 1: dirx=-1; diry=1; break;
                            case 2: dirx=-1; diry=-1; break;
                            case 3: dirx=1; diry=1; break;
                            case 4: dirx=1; diry=-1; break;
                            case 0: break;

                            }

                            if(direccion!=0){

                              #ifdef _WIN32
                                    Beep(500,50);
          #endif
                    if(accx==0 || accy==0) {accx=0.001; accy=0.001;}
```

```
                        accx=accx*(double)1.01; accy=accy*(double)1.03;
                        puntos++;
                        printf("\nPuntos: %d Anterior puntuacion: %d\n",puntos,antPuntos);
                       }
        }
        /*swap the graphics buffers*/
        if(velx<=MAX_VEL){
         velx+=accx/100.0;
        }
        if(vely<=MAX_VEL){
         vely+=accy/100.0;
        }

        cx+=velx*dirx;
        cy+=vely*diry;
        if((cx*arImXsize+(arImXsize*r))>=arImXsize && dirx==1) {dirx=-1;
           #ifdef _WIN32
            Beep(300,50);
      #endif
        }
        else if((cx*arImXsize-(arImXsize*r))<=0 && dirx==-1) {dirx=1;
           #ifdef _WIN32
            Beep(300,50);
      #endif
        }
        else if((cy*arImYsize+(arImYsize*r))>=arImYsize && diry==1) {diry=-1;
      #ifdef _WIN32
            Beep(300,50);
      #endif
        }
        else if((cy*arImYsize-(arImYsize*r))<=0) {cx=0.5;cy=0.5; accx=0; accy=0; velx=0.01,
vely=0.03; antPuntos=puntos; puntos=0;
      #ifdef _WIN32
                Beep(100,700);
      #endif
        }
        dibujaPelota();
        argSwapBuffers();
}

static void init( void )
{
        ARParam  wparam;
         srand( (unsigned int) time( NULL ) );
    velx=0.01;
         vely=0.03;
         accx=0;
         accy=0;
         dirx=1;diry=1;
         if(rand()%2<1) dirx=-1;
         if(rand()%2<1) diry=-1;
```

```
        cargarDeFichero();
        /* selecciona el color de borrado */
   glClearColor (0.0, 0.0, 0.0, 0.0);
   /* open the video path */
   if( arVideoOpen( vconf ) < 0 ) exit(0);
   /* find the size of the window */
   if( arVideoInqSize(&xsize, &ysize) < 0 ) exit(0);
   printf("Image size (x,y) = (%d,%d)\n", xsize, ysize);

   /* set the initial camera parameters */
   if( arParamLoad(cparam_name, 1, &wparam) < 0 ) {
      printf("Camera parameter load error !!\n");
      exit(0);
   }
   arParamChangeSize( &wparam, xsize, ysize, &cparam );
   arInitCparam( &cparam );
   printf("*** Camera Parameter ***\n");
   arParamDisp( &cparam );



   /* open the graphics window */
   argInit( &cparam, 2.0, 0, 0, 0, 0 );

        glEnable(GL_TEXTURE_2D);
     #ifdef _WIN32
      if(!cargarTGA("images\\pelota.tga", &texturas[0]) || !cargarTGA("images\\jugador.tga",
&texturas[1])) {
     #else
      if(!cargarTGA("images/pelota.tga", &texturas[0]) || !cargarTGA("images/jugador.tga",
&texturas[1])) {
     #endif
            printf("Error cargando textura\n");
                exit(1);
                }


}

/* cleanup function called when program exits */
static void cleanup(void)
{
        arVideoCapStop();
   arVideoClose();
   argCleanup();
}

/* draw the the AR objects */



static int cargarDeFichero(void) {
 FILE *fich;
```

```
 char c;
 fich=fopen("conf.con","r");

 if(fich != NULL) {

          fscanf(fich,"%d",&umbral);
          fscanf(fich," %c",&c);
    printf("%c",c);

          if(c=='#'){  //si el separador es correcto y no es fin de fichero carga valores;
           fscanf(fich,"%d",&minH);
           fscanf(fich,"%d",&maxH);
           fscanf(fich,"%d",&minS);
           fscanf(fich,"%d",&maxS);
           fscanf(fich,"%d",&minB);
           fscanf(fich,"%d",&maxB);
          }
  fclose(fich);
  return 1;
 }
 return 0;

}

static int guardarEnFichero(void) {
 FILE *fich;

 fich=fopen("conf.con","w");

 if(fich != NULL) {
  fprintf(fich,"%d\n#\n%d %d\n%d %d\n%d %d\n-
\n",umbral,minH,maxH,minS,maxS,minB,maxB);
  fclose(fich);
  return 1;
 }
 return 0;

}

static void dibujaPelota() {


/* inicializa los valores de la vista */
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluOrtho2D(0.0, 1.0, 0.0, 1.0);
glBindTexture(GL_TEXTURE_2D,texturas[0].ID);

//En lugar de dibujar un circulo, dibujamos un cuadrado y le pegamos la textura de la pelota, como
tiene un canal alpha para transparencias parece un circulo
glBegin(GL_POLYGON);
  glTexCoord2f(0.0,0.0); glVertex2f(cx-r,cy-r);
  glTexCoord2f(0.0,1.0); glVertex2f(cx-r,cy+r);
```

```
  glTexCoord2f(1.0,1.0); glVertex2f(cx+r,cy+r);
  glTexCoord2f(1.0,0.0); glVertex2f(cx+r,cy-r);

glEnd();
glFlush();

/* Vacia el buffer de dibujo */
glFlush ();

}

static void dibujaJugador(double x, double y) {

/* inicializa los valores de la vista */
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluOrtho2D(0.0, 1.0, 0.0, 1.0);

glBindTexture(GL_TEXTURE_2D,texturas[1].ID);

glBegin(GL_POLYGON);
//glRectd(x-PLAYER_WIDTH/2,y-
PLAYER_HEIGHT/2,x+PLAYER_WIDTH/2,y+PLAYER_HEIGHT/2);
  glTexCoord2f(0.0,0.0); glVertex2f(x-PLAYER_WIDTH/2,y-PLAYER_HEIGHT/2);
  glTexCoord2f(0.0,1.0); glVertex2f(x-PLAYER_WIDTH/2,y+PLAYER_HEIGHT/2);
  glTexCoord2f(1.0,1.0); glVertex2f(x+PLAYER_WIDTH/2,y+PLAYER_HEIGHT/2);
  glTexCoord2f(1.0,0.0); glVertex2f(x+PLAYER_WIDTH/2,y-PLAYER_HEIGHT/2);

glEnd();

/* Vacia el buffer de dibujo */
glFlush ();



}

static int colisionJugador(double jx, double jy, double px, double py, double prad) {

        double compx;
        double compy;
   jx*=100;
        jy*=100;
        px*=100;
        py*=100;
        prad*=100;
        compx = abs((int)px-(int)jx);
        compy = abs((int)py-(int)jy);

        if(compx <= 2*prad+PLAYER_WIDTH*100/2 && compy <=
2*prad+PLAYER_HEIGHT*100/2){
    //1 colision por la derecha-arriba
         //2 colision derecha-abajo
```

```
        //3 colision izquierda-arriba
        //4 colision izquierda-abajo

                if(px>=jx && py>=jy) return 1;
                else if(px>=jx && py<jy) return 2;
                else if(px<jx && py>=jy) return 3;
                else  return 4;
        }
        //0 no colision
        return 0;

}
```

Fichero arPuzzle.c

```
#ifdef _WIN32
#  include <windows.h>
#endif
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>

#ifndef __APPLE__
#  include <GL/glut.h>
#else
#  include <GLUT/glut.h>
#endif
#include <AR/gsub.h>
#include <AR/param.h>
#include <AR/ar.h>
#include <AR/video.h>
#include <formatopixel.h>
#include <time.h>




#define MIN_H 0
#define MAX_H 0
#define MIN_S 0
#define MAX_S 0
#define MIN_B 0
#define MAX_B 0

#define MAX_VEL 3*r
#define MAX_ACC 0.007
#define UMBRAL 5


int             xsize, ysize;
int                             thresh = 100;
int             count = 0;

//centro y radio del circulo
double cx=0.5;
double cy=0.5;
double velx;
double vely;
int dirx;
int diry;
double accx, accy;
double r=0.02;
int dentro = 0;
int punto[2];
int click;
```

```
long timer;
int          minH = MIN_H, maxH = MAX_H, minS = MIN_S, maxS = MAX_S, minB = MIN_B,
maxB = MAX_B, umbral = UMBRAL;
int invert = 0;
ARUint8        *dataPtr;


/* set video capture configuration */
#ifdef _WIN32
char                    *vconf = "flipV,flipH,showDlg"; // see video.h for a list of supported
parameters
#else
char                    vconf[256];
#endif

char          *cparam_name    = "Data/camera_para.dat";
ARParam        cparam;


typedef struct {
   GLubyte  *dibujo;       // Un puntero a los datos de la imagen
        GLuint          bpp;              // bpp significa bits per pixel (bits por punto) es la calidad en
palabras sensillas
        GLuint largo;           // Largo de la textura
        GLuint ancho;                   // Ancho de la textura
        GLuint ID;              // ID de la textura, es como su nombre para opengl
}textura;


typedef struct {
  double posx;
  double posy;
  double tamx;
  double tamy;
  int arrastrar;
  textura text;
  textura mask;
} objetoDibujable;

typedef objetoDibujable pieza;
typedef objetoDibujable pointer;

pieza piezas[4];
textura texturas[7];
textura mascaras[3];
pointer puntero;

static void   init(void);
static void   cleanup(void);
static void   keyEvent( unsigned char key, int x, int y);
static void   mainLoop(void);
static int    guardarEnFichero(void);
static int    cargarDeFichero(void);
```

```
static void creaPiezas();
static void dibujaPiezas(int pos);
static void creaPuntero();
static void dibujaPuntero();
static int comprobarColision();
static int comprobarPiezas(int pos);
int cargarTGA( char *nombre, textura *imagen) ;
void cambiarTamano(int largo, int ancho);




int cargarTGA( char *nombre, textura *imagen) {
	GLubyte		cabezeraTGA[12]={0,0,2,0,0,0,0,0,0,0,0,0};		// Cabezera de un tga sin compresion
   GLubyte	compararTGA[12];				// Aca vamos a comprar la cabezera
	GLubyte		cabezera[6];					// Los 6 bytes importantes
	GLuint		bytesporpunto;					// Cuantos bytes hay por punto
	GLuint		tamanoimagen;					// Aca guardaremos el tamaï¿½ de la imagen
	GLuint		temp,i;						// Variable temporal, y una para usar con el for
	GLuint		tipo=GL_RGBA;					// Nuestro typo por defecto, lo veremos mas abajo
	FILE *archivo=fopen(nombre, "rb");		// Cargamos nuestro archivo en memoria
	if( archivo == NULL ||				// Existe nuestro archivo??
	fread(compararTGA,1,sizeof(compararTGA),archivo)!=sizeof(compararTGA)  ||	// Hay 12 bytes para leer??
	   memcmp(cabezeraTGA,compararTGA,sizeof(compararTGA))!=0		||	// Son iguales??
	   fread(cabezera,1,sizeof(cabezera),archivo)!=sizeof(cabezera)) {
		if(archivo==NULL) {
			printf("No se encontro el archivo %s\n",nombre);
			return 0;		// No se abrio el archivo
		}
		else {
			fclose(archivo);
			return 0;
		}
	}
	/* Ahora hay que leer la cabezera del archivo, para saber cuanto es el largo, ancho, y los bytes por puntos,
	para eso aca hay una ilustracion de la cabezera :
	6 bytes -> xxxxxxx xxxxxxx xxxxxxx xxxxxxx xxxxxxx xxxxxxx
		|--- Largo ---| |---Ancho-----| |-bpp-|
	El dato del largo se guarda en el cabezera[0] y cabezera[1], para leerlo hay que multiplicar cabezera[0] por 256 y	sumarselo a cabezera[1], para leer ancho y bpp es el mismo procedimiento */
	imagen->largo=256*cabezera[1]+ cabezera[0];
	imagen->ancho=256*cabezera[3]+ cabezera[2];
```

```
        /* Ahora vemos si hay datos no validos, como largo o ancho iguales menores a 0 o iguales
a 0 */
        if( imagen->largo <= 0 ||       // Largo mayor que 0??
           imagen->ancho <= 0 ||       // Ancho mayor que 0??
           (cabezera[4]!=24 && cabezera[4]!=32)) {          // bpp es 24 o 32?? (solo se cargan 24 y
32 bpp)
            printf("Daton invalidos\n");
                fclose(archivo);
                return 0;
        }
        imagen->bpp=cabezera[4];    // Aca se guardan los bits por punto
        bytesporpunto=cabezera[4]/8;          // Aca los bytes por punto (1 byte = 8 bits)
      tamanoimagen=imagen->largo * imagen->ancho * bytesporpunto;      // Esta es la memoria
que nesecitaremos para guardar los datos de la textura
        /*Ahora reservamos espacio en memoria para nuestra textura, luego leemos la textura del
archivo */
        imagen->dibujo = (GLubyte *)malloc(tamanoimagen);       // Reservamos la memoria
necesaria para nuestra textura
        if(imagen->dibujo== NULL ||                     // Se logro reservar la memoria???
          fread(imagen->dibujo, 1, tamanoimagen, archivo) != tamanoimagen ) { // Se lee, y se
comprueba que lo leido es de la misma longitud que la asignada a a dibujo.
                if(imagen->dibujo != NULL) {
                        printf("Error leyendo imagen\n");
                        free(imagen->dibujo);
                } else printf("Error asignando memoria\n");
                fclose(archivo);
                return 0;
        }
        /* El formato tga guarda las imagenes en BGR, y opengl usa RGB,por lo cambiamos de
lugares */
      for(i=0; i< (int)tamanoimagen; i+=bytesporpunto)
        {
                temp=imagen->dibujo[i];                                 // Guardamos el primer valor
                imagen->dibujo[i] = imagen->dibujo[i + 2];          // Asignamos el nuevo primer
valor
                imagen->dibujo[i + 2] = temp;                           // Asignamos el ultimo valor
        }

        fclose (archivo);        // Cerramos el archivo

        /* Listo, terminamos con el codigo de carga, volvemos a opengl, ahora hay que asignarle a
la textura un ID, luego          decirle a opengl cuales son el largo, el ancho y los bpp */

      glGenTextures( 1 , &imagen[0].ID);    // Crea un ID para la textura, buscando un id que este
vacio
      glBindTexture(GL_TEXTURE_2D, imagen[0].ID);       // Seleccionamos nuestra textura
        if(imagen->bpp ==24) tipo= GL_RGB;       // Si nuestra textura es de 24 bits, entonces se
crea una textura rgb, sino una rgba
        /* Ahora creamos nuestra textura, entrando el largo, ancho y tipo */
      glTexImage2D(GL_TEXTURE_2D, 0, tipo, imagen[0].ancho, imagen[0].largo, 0, tipo,
GL_UNSIGNED_BYTE, imagen[0].dibujo);
        /* Ahora le decimos a opengl como queremos que se vea nuestra textura, MAG_FILTER es
cuando la textura es mas grande que el lugar donde la asignamos, y MIG_FILTER, es cuando la
```

```
textura es mas pequeï¿½ que el lugar donde la asignamos, GL_LINEAR es para que se vea bien
tanto cerca como lejos, pero ocupa bastante procesador. Otra opcion el GL_NEARES, que ocupa
menos prosesador */
        glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
        glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
        return 1;         // Todo salio bien
}


void cambiarTamano(int largo, int ancho) {
      if(ancho==0) ancho=1;          // Previene que dividamos por 0
      glMatrixMode(GL_PROJECTION);    // Escojemos la matriz de proyeccion
      glLoadIdentity();              // Se resetea la matriz
      glViewport(0,0,largo, ancho);   // Se va a usar toda la ventana para mostrar graficos
      gluPerspective( 45 ,           // Angulo de vision
                  (float)largo/(float)ancho,
                  1.0,             // Cuan cerca se puede ver
                  1000);           // Cuan lejos se puede ver
      glMatrixMode(GL_MODELVIEW);     // Escojemos la matriz de vista
         glLoadIdentity();                 // Se resetea la matriz
      gluLookAt( 0.0, 0.0, 0.0,       // Hacia donde miramos
              0.0, 0.0, -3.0,      // Desde donde miramos
              0.0, 1.0, 0.0);      // Que eje es el que esta hacia arriba
}



int main(int argc, char** args)
{
        //initialize applications


#ifndef _WIN32
char dev[256];
strcpy(vconf,"-dev=camara -channel=0 -width=320 -height=240 -palette=YUV420P");


if(argc==2) {
 strcpy(vconf,"-channel=0 -palette=YUV420P -width=320 -height=240 -dev=");
 strcat(vconf, args[1]);
}
else {
 FILE *f;
 f=fopen("camara","r");
 if(f == NULL) {
  printf("No se ha encontrado un enlace \"./camara\" al dispositivo de video. Por favor, indique el
nombre de dispositivo (normalmente /dev/video0 o /dev/video1) en el que se encuentra su camara:
");
  scanf("%s",&dev);
  strcpy(vconf, "-channel=0 -palette=YUV420P -width=320 -height=240 -dev=");
  strcat(vconf, dev);
```

```
 }
 else {
 fclose(f);
 }

}
#endif


   init();

        //start video capture
        arVideoCapStart();

        //start the main event loop
   argMainLoop( NULL, keyEvent, mainLoop );

        return 0;
}

static void   keyEvent( unsigned char key, int x, int y)
{    char resp;
   /* quit if the ESC key is pressed */
   if( key == 0x1b ) {
      printf("*** %f (frame/sec)\n", (double)count/arUtilTimer());
      cleanup();
      exit(0);
   }



        if( key == 'h' ) {
                printf("\nminH = %d, maxH = %d\n",minH,maxH);
           printf("\nIntroduce el valor minimo de H(tono):\n");
                scanf("%d",&minH);
           printf("\nIntroduce el valor maximo de H(tono):\n");
                scanf("%d",&maxH);

        }

   if( key == 'b' ) {
           printf("\nminB = %d, maxB = %d\n",minB,maxB);
           printf("\nIntroduce el valor minimo de B(brillo):\n");
                scanf("%d",&minB);
           printf("\nIntroduce el valor maximo de B(brillo):\n");
                scanf("%d",&maxB);
        }

   if( key == 's' ) {
                printf("\nminS = %d, maxS = %d\n",minS,maxS);
           printf("\nIntroduce el valor minimo de S(Saturacion):\n");
                scanf("%d",&minS);
           printf("\nIntroduce el valor maximo de S(Saturacion):\n");
```

```
                scanf("%d",&maxS);

        }


   if( key == 'r' ) {
        cx=0.5;
        cy=0.5;
        srand( (unsigned int) time( NULL ) );
    velx=0.01;
        vely=0.02;
        accx=0;
        accy=0;
        dirx=1;diry=1;
        if(rand()%2<1) dirx=-1;
        if(rand()%2<1) diry=-1;

        }



        if( key == 'i' ) {

                if(invert) invert = 0;
                 else invert = 1;
            printf("\nInversion realizada\n");

        }

        if( key == 'u' ) {

            printf("\nIntroduce el umbral:\n");
                scanf("%d",&umbral);

        }

   if( key == 'g' ) {

            printf("\nValores actuales: \nH: [%d-%d]\nS: [%d-%d]\nB: [%d-%d]
\n",minH,maxH,minS,maxS,minB,maxB);
            printf("\nDesea guardar los datos de configuracion de la sesion actual? (s/n)\n");
                scanf("%c",&resp);
                if(resp=='s') {
       if(guardarEnFichero()) printf("\nConfiguracion guardada en fichero.\n");
                 else {printf("\nError!. No se pudo guardar la configuracion en el fichero.\n");}
                }


        }

}
```

```
/* main loop */
static void mainLoop(void)
{

//    ARMarkerInfo    *marker_info;
          ARMarkerInfo2   *marker2;
          int             marker_num2;
    //int           marker_num;
    int           k;
          ARInt16         *limage;
    int           label_num;
    int           *area, *clip, *label_ref;
    double          *pos;
          int                                   seleccionada;



    marker_num2 = 0;


    /* grab a video frame */
    if( (dataPtr = (ARUint8 *)arVideoGetImage()) == NULL ) {
       arUtilSleep(2);
       return;
    }

    if( count == 0 ) arUtilTimerReset();
    count++;


          /* capture the next video frame */
          arVideoCapNext();
    glClear (GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
          glColor3f( 1.0, 0.0, 0.0 );
          glLineWidth(6.0);


    limage = arLabelingHSB( dataPtr, thresh,
                      &label_num, &area, &pos, &clip, &label_ref, 1, minH, maxH, minS, maxS, minB,
maxB, invert);
    if( limage == 0 )   {
              cleanup();
              exit(0);

          }

    marker2 = arDetectMarker3( limage, label_num, label_ref,
                               area, pos, clip, AR_AREA_MAX, AR_AREA_MIN, &marker_num2);
    if( marker2 == 0 ) {

   cleanup();
              exit(0);
```

```
	}

	dentro=0;

	if(timer==0 && marker_num2==0) {
	 click=0;
	 timer = (long) time(NULL);
	}
	else if (timer>0 && marker_num2>0){
		if((time(NULL)-timer)<=2){click=1;}
	 timer = 0;
	}
	else {
	 click=0;
	}


		seleccionada = comprobarColision();
		if(seleccionada){
			if(click) {
				if(!piezas[seleccionada-1].arrastrar && !puntero.arrastrar) {
				 piezas[seleccionada-1].arrastrar = 1;
				 puntero.arrastrar = 1;

				}
				else if(piezas[seleccionada-1].arrastrar) {
				 piezas[seleccionada-1].arrastrar = 0;
				 puntero.arrastrar = 0;

				}

			}//end_if

		}//end_if


	for(k=0;puntero.arrastrar && k<4;k++){
    if(piezas[k].arrastrar==1) { piezas[k].posx=(double)marker2[0].pos[0]/(double)arImXsize;
piezas[k].posy=1.0-((double)marker2[0].pos[1]/(double)arImYsize);}

	}

	if(marker_num2>0) {
	 puntero.posx=marker2[0].pos[0]/(double)arImXsize;
	 puntero.posy=1-marker2[0].pos[1]/(double)arImYsize;
	}

	if(puntero.arrastrar) {
		puntero.text = texturas[6];
		puntero.mask = mascaras[2];
	}
	else if(seleccionada){
		puntero.text = texturas[5];
```

```
                puntero.mask = mascaras[1];
        }
        else {
                puntero.text = texturas[4];
                puntero.mask = mascaras[0];
        }

        if (comprobarPiezas(seleccionada-1)) {
         printf("Correcto!");
        }
        dibujaPiezas(seleccionada-1);
        dibujaPuntero();
        argSwapBuffers();

}

static void init( void )
{
        ARParam  wparam;
         srand( (unsigned int) time( NULL ) );
    velx=0.01;
         vely=0.02;
         accx=0;
         accy=0;
         dirx=1;diry=1;
         if(rand()%2<1) dirx=-1;
         if(rand()%2<1) diry=-1;

         click=0;
         timer=time(NULL);


        cargarDeFichero();
        /* selecciona el color de borrado */
   glClearColor (0.0, 0.0, 0.0, 0.0);
   /* open the video path */
   if( arVideoOpen( vconf ) < 0 ) exit(0);
   /* find the size of the window */
   if( arVideoInqSize(&xsize, &ysize) < 0 ) exit(0);
   printf("Image size (x,y) = (%d,%d)\n", xsize, ysize);

   /* set the initial camera parameters */
   if( arParamLoad(cparam_name, 1, &wparam) < 0 ) {
      printf("Camera parameter load error !!\n");
      exit(0);
   }
   arParamChangeSize( &wparam, xsize, ysize, &cparam );
   arInitCparam( &cparam );
   printf("*** Camera Parameter ***\n");
   arParamDisp( &cparam );

   /* open the graphics window */
   argInit( &cparam, 2.0, 0, 0, 0, 0 );
```

```
   glClearColor(0.0, 0.0, 0.0, 0.0);       // Pondremos aca la funcion glclearcolor
   glClearDepth(1.0);
   glEnable(GL_DEPTH_TEST);
   glDepthFunc(GL_LEQUAL);
   glHint(GL_PERSPECTIVE_CORRECTION_HINT, GL_NICEST);      // Esto hace que opengl
calcule las perspectivas de la mejor forma, quita un poco de rendimiento, pero hace que las
perspectivas se vean un poco mejor.
         glEnable(GL_TEXTURE_2D);

#ifdef _WIN32
      if(!cargarTGA("images\\me1.tga", &texturas[0]) || !cargarTGA("images\\me2.tga", &texturas
[1]) || !cargarTGA("images\\me3.tga", &texturas[2]) || !cargarTGA("images\\me4.tga", &texturas
[3]) || !cargarTGA("images\\puntflecha.tga", &texturas[4]) || !cargarTGA("images\\puntmano.tga",
&texturas[5]) || !cargarTGA("images\\puntcogiendo.tga", &texturas[6]) || !cargarTGA
("images\\puntflecha-mask.tga", &mascaras[0]) || !cargarTGA("images\\puntmano-mask.tga",
&mascaras[1]) || !cargarTGA("images\\puntcogiendo-mask.tga", &mascaras[2])) {
#else
      if(!cargarTGA("images/me1.tga", &texturas[0]) || !cargarTGA("images/me2.tga", &texturas
[1]) || !cargarTGA("images/me3.tga", &texturas[2]) || !cargarTGA("images/me4.tga", &texturas[3])
|| !cargarTGA("images/puntflecha.tga", &texturas[4]) || !cargarTGA("images/puntmano.tga",
&texturas[5]) || !cargarTGA("images/puntcogiendo.tga", &texturas[6]) || !cargarTGA
("images/puntflecha-mask.tga", &mascaras[0]) || !cargarTGA("images/puntmano-mask.tga",
&mascaras[1]) || !cargarTGA("images/puntcogiendo-mask.tga", &mascaras[2])) {

#endif
      printf("Error cargando imagenes\n");
                  exit(1); // Cargamos la textura y chequeamos por errores
                  }

         //creamos las piezas y el puntero
   creaPuntero();
         creaPiezas();


}

/* cleanup function called when program exits */
static void cleanup(void)
{
         arVideoCapStop();
   arVideoClose();
   argCleanup();
}

/* draw the the AR objects */



static int cargarDeFichero(void) {
 FILE *fich;
 char c;
```

```
 fich=fopen("conf.con","r");

 if(fich != NULL) {

          fscanf(fich,"%d",&umbral);
          fscanf(fich," %c",&c);
    printf("%c",c);

          if(c=='#'){  //si el separador es correcto y no es fin de fichero carga valores;
           fscanf(fich,"%d",&minH);
           fscanf(fich,"%d",&maxH);
           fscanf(fich,"%d",&minS);
           fscanf(fich,"%d",&maxS);
           fscanf(fich,"%d",&minB);
           fscanf(fich,"%d",&maxB);
          }
  fclose(fich);
  return 1;
 }

 return 0;

}

static int guardarEnFichero(void) {
 FILE *fich;

 fich=fopen("conf.con","w");

 if(fich != NULL) {
  fprintf(fich,"%d\n#\n%d %d\n%d %d\n%d %d\n-
\n",umbral,minH,maxH,minS,maxS,minB,maxB);
  fclose(fich);
  return 1;
 }
 return 0;

}



static void creaPiezas() {

          piezas[0].posx = 0.6;
          piezas[0].posy = 0.7;
          piezas[0].tamx = 0.2;
          piezas[0].tamy = 0.2;
          piezas[0].arrastrar = 0;
          piezas[0].text = texturas[0];
          piezas[1].posx = 0.4;
          piezas[1].posy = 0.4;
          piezas[1].tamx = 0.2;
          piezas[1].tamy = 0.2;
```

```
        piezas[1].arrastrar = 0;
        piezas[1].text = texturas[1];
        piezas[2].posx = 0.7;
        piezas[2].posy = 0.2;
        piezas[2].tamx = 0.2;
        piezas[2].tamy = 0.2;
        piezas[2].text = texturas[2];
        piezas[2].arrastrar = 0;
        piezas[3].posx = 0.2;
        piezas[3].posy = 0.7;
        piezas[3].tamx = 0.2;
        piezas[3].tamy = 0.2;
        piezas[3].text = texturas[3];
        piezas[3].arrastrar = 0;

}

static void dibujaPiezas(int pos) {

int i;
float z;

/* inicializa los valores de la vista */
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluOrtho2D(0.0,1.0,0.0,1.0);

glColor4f(0.0,0.0,0.0,1.0);


//Las piezas se dibujan a partir de su centro (posx, posy) y conociendo el tamaï¿½ del lado en X y en Y, sumando y restando alternativamente la mitad del tamaï¿½ X y tamaï¿½ Y a las coordenadas X,Y del centro se obtienen las cuatro esquinas
for(i=0;i<4;i++) {

  if (piezas[i].arrastrar) z=0.5; else z=0; //si la pieza esta siendo arrastrada se dibuja encima del resto, para ello le asignamos z=0.5 para que este mas cerca de la camara y a las demas z=0
  glBindTexture(GL_TEXTURE_2D,piezas[i].text.ID);
  glBegin(GL_POLYGON);
  glTexCoord2f(1.0,1.0); glVertex3f(piezas[i].posx-piezas[i].tamx/2,piezas[i].posy-piezas[i].tamy/2,z);
  glTexCoord2f(1.0,0.0); glVertex3f(piezas[i].posx-piezas[i].tamx/2,piezas[i].posy+piezas[i].tamy/2,z);
  glTexCoord2f(0.0,0.0); glVertex3f(piezas[i].posx+piezas[i].tamx/2,piezas[i].posy+piezas[i].tamy/2,z);
  glTexCoord2f(0.0,1.0); glVertex3f(piezas[i].posx+piezas[i].tamx/2,piezas[i].posy-piezas[i].tamy/2,z);

  glEnd();
}
/* Vacia el buffer de dibujo */
glFlush();
```

```
}

static void creaPuntero() {
	puntero.posx = 0.5;
	puntero.posy = 0.5;
	puntero.tamx = 0.05;
	puntero.tamy = 0.1;
	puntero.arrastrar = 0;
	puntero.text = texturas[4];
	puntero.mask = mascaras[0];
}

static void dibujaPuntero() {

 glMatrixMode(GL_PROJECTION);
 glLoadIdentity();
 gluOrtho2D(0.0, 1.0, 0.0, 1.0);

 glEnable(GL_BLEND);
 glBindTexture(GL_TEXTURE_2D,puntero.mask.ID);
 glBlendFunc(GL_DST_COLOR,GL_ZERO);

 glBegin(GL_POLYGON);
 //la coordenada z sera 1 para que sea el puntero siempre este por encima de los demas objetos, la
pieza seleccionada estara a 0.5 y las demas a 0.
  glTexCoord2f(1.0,1.0); glVertex3f(puntero.posx-puntero.tamx/2,puntero.posy-puntero.tamy/2,1);
  glTexCoord2f(1.0,0.0); glVertex3f(puntero.posx-
puntero.tamx/2,puntero.posy+puntero.tamy/2,1);
  glTexCoord2f(0.0,0.0); glVertex3f
(puntero.posx+puntero.tamx/2,puntero.posy+puntero.tamy/2,1);
  glTexCoord2f(0.0,1.0); glVertex3f(puntero.posx+puntero.tamx/2,puntero.posy-
puntero.tamy/2,1);
glEnd();

glBlendFunc(GL_ONE,GL_ONE);
glBindTexture(GL_TEXTURE_2D,puntero.text.ID);
 glBegin(GL_POLYGON);
  glTexCoord2f(1.0,1.0); glVertex3f(puntero.posx-puntero.tamx/2,puntero.posy-puntero.tamy/2,1);
  glTexCoord2f(1.0,0.0); glVertex3f(puntero.posx-
puntero.tamx/2,puntero.posy+puntero.tamy/2,1);
  glTexCoord2f(0.0,0.0); glVertex3f
(puntero.posx+puntero.tamx/2,puntero.posy+puntero.tamy/2,1);
  glTexCoord2f(0.0,1.0); glVertex3f(puntero.posx+puntero.tamx/2,puntero.posy-
puntero.tamy/2,1);
glEnd();

glDisable(GL_BLEND);

}

static int comprobarPiezas(int pos) {
 int i;
 double prefx, prefy;
```

```
//efecto iman, las piezas se "pegan" cuando se encuentran suficientemente cerca de otra pieza

for(i=0;i<4;i++) {
 if(i!=pos && (piezas[pos].posx+piezas[pos].tamx/2>=piezas[i].posx-piezas[i].tamx/2-0.025) &&
(piezas[pos].posx<piezas[i].posx-piezas[i].tamx)){
//         if(i!=pos && ((piezas[pos].posy+piezas[pos].tamy/2>=piezas[i].posy+piezas[i].tamy/2-
0.025) && piezas[pos].posy<piezas[i].posy+piezas[i].tamy) || ((piezas[i].posy+piezas[i].
tamy/2>=piezas[pos].posy+piezas[pos].tamy/2-0.025) && piezas[i].posy<piezas[pos].posy+piezas
[pos].tamy)){
  if(i!=pos && ((piezas[pos].posy>=piezas[i].posy-0.025) && piezas[pos].posy<piezas[i].posy) ||
((piezas[i].posy>=piezas[pos].posy-0.025) && piezas[i].posy<piezas[pos].posy)){
           piezas[pos].posx=piezas[i].posx-piezas[i].tamx;
           piezas[pos].posy-=piezas[pos].posy-piezas[i].posy;
          }//end_if
}//end_if
 else if(i!=pos && (piezas[i].posx+piezas[i].tamx/2>=piezas[pos].posx-piezas[pos].tamx/2-0.025)
&& (piezas[i].posx<piezas[pos].posx-piezas[pos].tamx)){
  if(i!=pos && ((piezas[pos].posy>=piezas[i].posy-0.025) && piezas[pos].posy<piezas[i].posy) ||
((piezas[i].posy>=piezas[pos].posy-0.025) && piezas[i].posy<piezas[pos].posy)){
           piezas[pos].posx=piezas[i].posx+piezas[i].tamx;
           piezas[pos].posy-=piezas[pos].posy-piezas[i].posy;
          }//end_if
 }//end_else

 else if(i!=pos && (piezas[i].posy+piezas[i].tamy/2>=piezas[pos].posy-piezas[pos].tamy/2-0.025)
&& (piezas[i].posy<piezas[pos].posy-piezas[pos].tamy)){
          if(i!=pos && ((piezas[pos].posx>=piezas[i].posx-0.025) && piezas[pos].posx<piezas[i].
posx) || ((piezas[i].posx>=piezas[pos].posx-0.025) && piezas[i].posx<piezas[pos].posx)){

 piezas[pos].posy=piezas[i].posy+piezas[i].tamy;
           piezas[pos].posx-=piezas[pos].posx-piezas[i].posx;
          }//end_if

 }//end_else

 else if(i!=pos && (piezas[pos].posy+piezas[pos].tamy/2>=piezas[i].posy-piezas[i].tamy/2-0.025)
&& (piezas[pos].posy<piezas[i].posy-piezas[i].tamy)){
          if(i!=pos && ((piezas[pos].posx>=piezas[i].posx-0.025) && piezas[pos].posx<piezas[i].
posx) || ((piezas[i].posx>=piezas[pos].posx-0.025) && piezas[i].posx<piezas[pos].posx)){
           piezas[pos].posy=piezas[i].posy-piezas[i].tamy;
           piezas[pos].posx-=piezas[pos].posx-piezas[i].posx;
          }//end_if

 }//end_else

}//end_for

prefx=piezas[0].posx+piezas[0].tamx/2; //Establecemos un punto de referencia para comprobar la
posicion de las piezas
prefy=piezas[0].posy-piezas[0].tamy/2; //Este punto sera la esquina inferior derecha de la primera
pieza, ya que es el unico punto que comparten las 4 piezas
```

//Para saber si estan bien colocadas el resto de piezas respecto de la primera, se comprueba si la esquina correspondiente a cada pieza, coincide con el punto de referencia

```
/*

        _______
       |   |   |
       |_0_|_1_|
       |   |   |
       |_2_|_3_|

*/

 if((piezas[1].posx-piezas[1].tamx/2 == prefx && piezas[1].posy-piezas[1].tamy/2 == prefy)
            && (piezas[2].posx+piezas[2].tamx/2 == prefx && piezas[2].posy+piezas[2].tamy/2 ==
prefy)
            && (piezas[3].posx-piezas[3].tamx/2 == prefx && piezas[3].posy+piezas[3].tamy/2 ==
prefy))
 {
  return 1;
 }

 return 0;
}


static int comprobarColision() {

 int i;
 for(i=0;i<4;i++) {
  if((puntero.posx+puntero.tamx/2>=piezas[i].posx-piezas[i].tamx/2) && (puntero.posx-
puntero.tamx/2 <= piezas[i].posx+piezas[i].tamx/2) && (puntero.posy+puntero.tamy/2>=piezas[i].
posy-piezas[i].tamy/2) && (puntero.posy-puntero.tamy/2 <= piezas[i].posy+piezas[i].tamy/2)){
   return i+1;
  }
 }
  return 0;

}
```