
Escuela Politécnica Superior de Elche

# CONTROL DE ROBOTS Y SISTEMAS SENSORIALES

## PRÁCTICAS DE CONTROL VISUAL DE ROBOTS
## Taller 2. Captura y Procesamiento de Imágenes

**Prof. Luis Miguel Jiménez**

**Departamento de Ingeniería**
**Área de Ingeniería de Sistemas y Automática**

## 1 OBJETIVO

El objetivo de este documento es describir el proceso de captura y procesamiento de imágenes en la tarjeta controladora Eyebot. Se recomienda consultar el taller de introducción al Eyebot y el servidor web (http://lorca.umh.es/isa/es/temas/eyebot) donde se encuentra documentación adicional.

Se describirán brevemente la captura de imágenes, la transferencia de imágenes al PC y la librería de procesamiento de imágenes.

## 2 MATERIAL EMPLEADO

La práctica se realizará en grupos de tres personas, disponiendo de un PC con S.O. Windows 95/98/2000/XP, el entorno de desarrollo Robios 6.0, y un Robot SoccerBot. La documentación y el software se puede encontrar en el servidor http://lorca.umh.es/isa/es/temas/eyebot

El sensor del robot utilizado en este taller será la cámara EyeCam (figura 1)

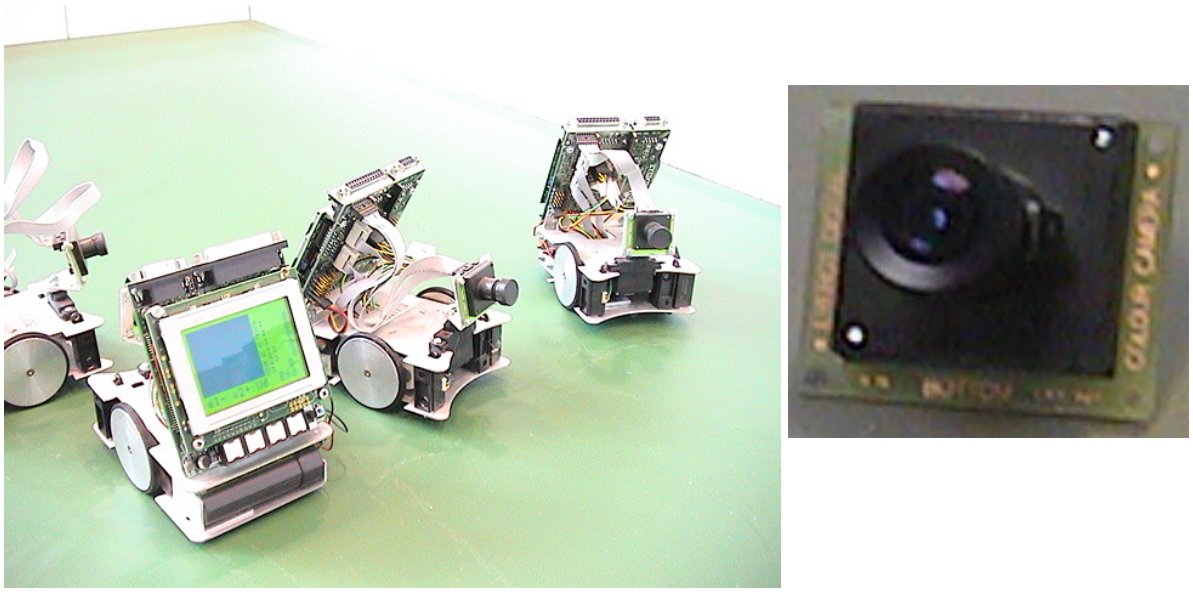

Figura 1. Robots SoccerBot disponibles en el departamento y cámara EyeCam

La cámara tiene una resolución de 80*60 pixels con 24 bits de profundidad de color (RGB). Permite una velocidad de captura entorno a los 5 fps. Dispone de enfoque manual mediante una rosca situada en la lente.

## 3 PROGRAMACIÓN DE LA CÁMARA EN ROBIOS

Para familiarizarnos con el manejo de la captura de imágenes vamos a crear un programa denominado ***camara.c*.**

En la tabla 1 se resumen las funciones principales involucradas. Existen funciones adicionales para la configuración los parámetros de la cámara, pero no todas son aplicables a la EyeCam. En las tablas 2, 3 y 4 se muestran los tipos de datos y las funciones básicas de procesamiento de imágenes integradas en el S.O.

**Camera**
The following functions handle initializing and image reading from either grayscale or color camera:

```
int CAMInit (int zoom);
        Input:          (zoom) zoom factor
                        Valid Values are: WIDE,NORMAL,TELE
        Output:         (returncode) Cameraversion or Errorcode
                        Valid values are:       255 = nocamera connected
                                                254 = camera init error
                                                0-15 = version of b/w camera
                                                16-31 = version of color camera
        Semantics:      Reset and initialize connected Quickcam

int CAMRelease (void);
        Input:          NONE
        Output:         (returncode) 0 on success, -1 on error.
        Semantics:      Release all resources allocated using CAMInit().

int CAMGetFrame ( image *buf);
        Input:          (buf) a pointer to a grey scale image
        Output:         NONE
        Semantics:      Read an image from grey scale camera.
                        NEW: Return 8 bit gray values 0 (black) .. 255 (white)

int CAMGetColFrame ( colimage *buf, int convert);
        Input:          (buf) a pointer to a color image
                        (convert) flag if image should be reduced to 4
                          bit on the fly
                                0 = get 24bit color image
                                1 = get 4bit greyscale image
        Output:         NONE
        Semantics:      Read an image from color cam and reduce it
                        eventually to 4 bit grey scale.  The colorimage
                        can be reduced to greyscale afterwards by
                        using IPColor2Grey(...)
                        HINT: buf should be a pointer to an 'image' if
                        conversion is enabled, like this:
                              image buffer;
                              CAMGetColFrame((colimage*)&buffer, 1);
```

**Tabla 1**. Librería ROBIOS para manejo de la cámara

Los tipos de datos usados en los parámetros de las funciones son:

```
Data Types:

        /* image is 80x60 but has a border of 1 pixel */
        #define imagecolumns 82
        #define imagerows 62

        typedef BYTE image[imagerows][imagecolumns];
        typedef BYTE colimage[imagerows][imagecoulmns][3];
```

**Tabla 2**. tipos de datos de la librería de imágnes

**Image Processing**
Basic image processing functions (library robios):

```
int IPLaplace (image *src, image *dest);
        Input:          (src) source b/w image
        Output:         (dest) destination b/w image
        Semantics:      The Laplace operator is applied to the source image
                        and the result ist written to the destination image

int IPSobel (image *src, image *dest);
        Input:          (src) source b/w image
        Output:         (dest) destination b/w image
        Semantics:      The Sobel operator is applied to the source image
                        and the result ist written to the destination image

int IPDither (image *src, image *dest);
        Input:          (src) source b/w image
        Output:         (dest) destination b/w image
        Semantics:      The Dithering operator with a 2x2 pattern is applied
                        to the source image and the result ist written to the
                        destination image

int IPDiffer (image *current, image *last, image *dest);
        Input:          (current) the current b/w image
                        (last) the last read b/w image
        Output:         (dest) destination b/w image
        Semantics:      Calculate the grey level difference at each pixel
                        position between current and last image, and
                        store the result in destination.

int IPColor2Grey (colimage *src, image *dest);
        Input:          (src) source color image
        Output:         (dest) destination b/w image
        Semantics:      Convert RGB color image given as source to 4-bit
                        grey level image and store the result in
                        destination.
```

**Tabla 2**. Librería ROBIOS para procesamiento de imágenes

Se puede encontrar una descripción más detallada de la funciones en el fichero (***image-processing.pdf***) disponible en el servidor web.

**Pantalla LCD**
Funciones básicas para mostrar imágenes en la pantalla LCD:

```
int LCDPutGraphic ( image *buf);
        Input:          (buf) pointer to a greyscale image (80*60 pixel)
        Output:         NONE
        Semantics:      Write the given graphic b/w to the display
                        it will be written starting in the top left corner
                        down to the menu line.  Only 80x54 pixels will
                        be written to the LCD, to avoid destroying the
                        menu line.

int LCDPutColorGraphic ( colimage *buf);
        Input:          (buf) pointer to a color image (80*60 pixel)
        Output:         NONE
        Semantics:      Write the given graphic b/w to the display
                        it will be written starting in the top left corner
                        down to the menu line.  Only 80x54 pixels will
                        be written to the LCD, to avoid destroying the
                        menu line.  Note: The current implementation
                        destroys the image content.
```

**Tabla 3**. Librería ROBIOS para mostrar imágenes en la pantalla LCD

**Ejemplo:**

El programa propuesto es muy sencillo y puede servir de punto de partida a una aplicación más compleja. Se declaran dos variables de tipo imagen: una en color y otra en nivel de gris. En primer lugar inicializa la cámara comprobando si existe algún error siguiendo el mecanismo descrito en la práctica anterior. Limpia y configura la pantalla activando una tecla en el menú. Por último ejecuta un bucle que captura una imagen en color, y la nuestra en pantalla hasta que se pulse la tecla 4. Por último se libera la cámara y el programa termina.

```
/* camara.c */
#include <eyebot.h>

colimage img;

int main()
{
       int cam;
       cam=CAMInit(NORMAL);
       if (cam==NOCAM  || cam== INITERROR)
       {
              OSError("Error en la cámara!\n", cam, FALSE);
              return -1;
       }
       LCDClear();
       LCDMode(SCROLLING | NOCURSOR);
       LCDMenu("","","","End");

       while (KEYRead()!=KEY4)
       {
              CAMGetColFrame(&img,FALSE);
              LCDPutColorGraphic(&img);
       }
       CAMRelease();
      return 0;
}
```

Compilaremos el programa y lo descargaremos en el robot. Una vez ejecutado iremos tomando imágenes ajustando el enfoque de la cámara. Si no disponemos del robot podemos utilizar el simulador que nos mostrará las imágenes simuladas en color.

Comentarios:

- La pantalla LCD solo permite visualizar dos niveles por lo que la imagen será poco descriptiva. Tratad de enfocar objetos oscuros sobre fondo claro.

## 4 TRANSFERENICA DE IMÁGENES AL PC

Como hemos comprobado la pantalla LCD no nos va a permitir la visualización de imágenes necesaria en la etapa de diseño en la que necesitamos enfocar y ajustar los parámetros de la aplicación.

Vamos a describir como conectar el robot al PC para poder visualizar de forma continua las imágenes en color en la pantalla del PC.

Utilizaremos una aplicación denominada ***EyeView*** que tiene dos partes:

1) *Parte ejecutada en el Robot*:
   En el directorio c*:\Eyebot\EyeView1.2* podemos localizar el programa ***eyeview.hex***. El transfiere imágenes tanto vía radio como por el puerto serie.

   Copiaremos el programa al subdirectorio *c:\eyebot\progs*. Transferiremos el programa ***eyeview.hex*** al robot y lo ejecutaremos. Podemos configurar la velocidad y el puerto serie (SERIAL1, 115200). Pulsaremos el botón OK y a continuación Con para la transmisión continua.

2) *Parte ejecutada en el PC:*
   En el directorio c*:\Eyebot\EyeView1.2* podemos localizar la aplicación denominada ***CameraViewer.exe*** . Antes de ejecutarla debemos editar el fichero ***configuration.ini*** . Este fichero debe contener los siguientes parámetros:

```
[Camera Information]
EyeBotSerial=COM2
EyeBotBaudrate=115200
EyeBotConnectionType=cable
# Options are: cable, radio, IRmate

[Plugins]
NumberOfPlugins=1
Plugin1=Plugins\EyeBot.cam

[Startup]
OpenWindows=1
Window1=EyeBot Camera
WindowScale=2
```

   Una vez configurado lo guardaremos y ejecutaremos la aplicación. La aplicación despliega dos ventanas, en una de ellas se mostrará la imagen que esté transmitiendo el robot por el puerto serie. Podemos almacenar la imagen en formato *jpg* pulsando en el botón superior izquierda. Esta imagen la podemos cargar en el programa **Títere** para el diseño de la aplicación de procesamiento de imágenes.


**Figura 2**. Imagen de la cámara capturada por Eyeview

## ENFOQUE DE LA IMÁGEN

La parte frontal de la cámara EyeCam tiene una lente enfocable mediante una rosca. Giraremos la rosca de enfoque hasta conseguir una imagen bien enfocada en el PC.

> **Nota**: *para algún despistado que esté obteniendo una imagen negra, recordar que la cámara lleva una tapa en la lente que deberemos quitar para tomar imágenes y volver a colocar cuando terminemos para proteger la lente.*

## 5 LA LIBRERÍA DE PROCESAMIENTO DE IMÁGENES

A parte de las funciones básicas de procesamiento de imágenes disponibles en ROBIOS se dispone de dos librerías adicionales con funciones más extensas. El uso de estas librerías requiere estudiar las funciones disponibles y los requisitos para su ejecución.

### *5.1 Librería Improc*:

La versión precompilada como librería se encuentra ubicada en el directorio *c:\Eyebot\robios60\libs* (fichero ***libimpro.a***) disponiendo de un ejemplo de uso en *cam.c*. Es preciso indicar en la compilación el uso de la librería libimpro.a así como la librería matemática (***gcc68 cam.c –o cam.hex –limpro -lm***). También está disponible el código fuente de esta librería.

***Nota***: esta librería no está disponible en el simulador EyeSim. En caso de necesitarla debemos compilar el código fuente (***gccsim cam.c impro.c –o cam.dll***)

Las tablas 4 y 5 describen las funciones disponibles y los tipos de datos declarados en improc.h (esta cabecera se incluye automáticamente dentro de *eyebot.h*, por lo que no es necesario especificarla en nuestro programa)

```
void IPL_init(IPL_CONFIG * config);
void IPL_laplace(BYTE * imageIn, BYTE * imageOut);
void IPL_sobel(BYTE * imageIn, BYTE * imageOut);
void IPL_mean(BYTE * imageIn, BYTE * imageOut);
void IPL_threshold(BYTE * imageIn, BYTE * imageOut, BYTE threshold);
void IPL_gray_stretch(BYTE * imageIn, BYTE * imageOut);
void IPL_gray_reduce(BYTE * imageIn, BYTE * imageOut, int numvals);
void IPL_gen_histogram (BYTE * imageIn, int *histogram);
void IPL_equal_histogram (BYTE * imageIn, BYTE * imageOut, int *histogram);
void IPL_erosion (BYTE * imageIn, BYTE * imageOut, char * struc);
void IPL_dilation (BYTE * imageIn, BYTE * imageOut, char * struc);
void IPL_open (BYTE * imageIn, BYTE * imageOut, char * struc);
void IPL_close (BYTE * imageIn, BYTE * imageOut, char * struc);
void IPL_fill (BYTE * imageIn, BYTE * imageOut, int x, int y, char * struc);
void IPL_connected(BYTE * imageIn, BYTE * imageOut, int x, int y, char * struc);
void IPL_boundary(BYTE * imageIn, BYTE * imageOut, char * struc);
void IPL_skeleton(BYTE * imageIn, BYTE * imageOut, char * struc);
void IPL_identity(BYTE * imageIn, BYTE * imageOut);
int  IPL_contour(BYTE * imageIn, BYTE * imageOut, IPLPoint * result, BYTE threshold);
void IPL_median(BYTE * imageIn, BYTE * imageOut);
void IPL_min(BYTE * imageIn, BYTE * imageOut);
void IPL_max(BYTE * imageIn, BYTE * imageOut);
void IPL_negation(BYTE * imageIn, BYTE * imageOut);
void IPL_difference(BYTE * image1, BYTE * image2, BYTE * imageOut);
void IPL_noise(BYTE * imageIn, BYTE * imageOut, double noise);
int  IPL_count(BYTE * imageIn, BYTE * imageOut, BYTE val);
int  IPL_count_nobound(BYTE * imageIn, BYTE * imageOut, BYTE val);
void IPL_corner(BYTE * imageIn, BYTE * imageOut);
void IPL_and(BYTE *imageIn1, BYTE * imageIn2, BYTE *imageOut);
void IPL_or(BYTE *imageIn1, BYTE * imageIn2, BYTE *imageOut);
int  IPL_equal(BYTE *imageIn1, BYTE *imageIn2);
int  IPL_equal_nobound(BYTE *image1, BYTE *image2);
void IPL_showxy(BYTE *imageIn, BYTE *imageOut, int x, int y);
void IPL_dither(BYTE *imageIn, BYTE *imageOut);
void IPL_overlay(BYTE *imageIn1, BYTE *imageIn2, BYTE *imageOut, BYTE gray_val);
void IPL_region_growing(BYTE *imageIn, BYTE *imageOut, int thresh);
int  IPL_FindCircles(BYTE * imageIn, BYTE * imageOut, XYDistance * result);
```

**Tabla 4**. Funciones de la librería Improc

```
typedef struct
{
   int imageWidth;
   int imageHeight;
   int colorBlack;
   int colorWhite;
} IPL_CONFIG;


typedef struct
{
   short x;
   short y;
} IPLPoint;

typedef struct
{
   double x;
   double y;
} XYDistance;
```

**Tabla 5**. Tipos de datos de la librería Improc

El uso no es muy complicado y solo requiere inicializar una serie de variables globales mediante la función **`IPL_init()`** tal como se describe en el ejemplo.

```
/** cam.c **/

#include <eyebot.h>
#include <libimpro/improc.h>

/** picture to work on */
colimage img;

/** picture for LCD-output */
image greyimg, borderimg, umbimg ;
int camera;
IPL_CONFIG _IPLConfig;

int initCamera(void);


int main()
{
        BYTE th = 90;
        initCamera();

        LCDClear();
        LCDMode(SCROLLING|NOCURSOR);
        LCDMenu("","","","End");
        while (KEYRead()!=KEY4)
        {
        CAMGetColFrame(&img,FALSE);
        IPColor2Grey(&img,&greyimg);
                IPSobel(&greyimg, &borderimg);
                IPL_threshold((BYTE *)&borderimg, (BYTE *)&umbimg, th);
                LCDPutGraphic(&borderimg);
        }

        CAMRelease();
        return 0;
}
```

```
/* ********************************************************************** */
/* This routine initializes the Image Processing system, the camera,     */
/* IPL library,  the screen constants
            */
/* ********************************************************************** */
int initCamera(void)
{

            /* Initialize the Image Processing System */

            _IPLConfig.colorBlack = 0;
            _IPLConfig.colorWhite = 15;
            _IPLConfig.imageWidth = imagecolumns;
            _IPLConfig.imageHeight = imagerows;

            IPL_init(&_IPLConfig);

            /* Initialize camera */
            camera=CAMInit(WIDE);
            if (camera==NOCAM)
                        OSError("No camera!\n", camera, FALSE);
            else if (camera==INITERROR)
            OSError("CAMInit!\n", camera, FALSE);

            return(0);
}
```

### *5.1 Librería LibVision*:

Se encuentra ubicada en el directorio c*:\Eyebot\examples60rob\vision\libvision-2.2* (ficheros ***LibVision.c, LibVision.h***). Se dispone también de una librería ya compilada en (*LibVision.a*) con lo que basta con enlazar con ella nuestra aplicación (se debe revisar el fichero *makefile* de ejemplo para ver los parámetros de enlazado de la librería).

Nota: como en el caso anterior esta librería no está disponible en el simulador EyeSim. En caso de necesitarla debemos compilar el código fuente

Se trata de una librería para el procesamiento de imágenes en color que se desarrollo para en concurso en la RoboCup por lo que se pueden encontrar funciones interesantes. En el directorio indicado se dispone de la documentación de las funciones tanto en formato html como en formato pdf ( *LibVision2.html, LibVision2.pdf*)

En el subdirectorio test se encuentra un ejemplo de utilización.

| **Funciones Librería LibVision** | |
|---|---|
| 1. VIS_RGB2Hue | 19. VIS_ColFind |
| 2. VIS_RGB2Sat | 20. VIS_CamCal |
| 3. VIS_RGB2Int | 21. VIS_ModifyCam |
| 4. VIS_FillHueTable | 22. VIS_ModifyAlgo |
| 5. VIS_FillRGBSpace | 23. VIS_InitDistance |
| 6. VIS_FillWhiteClass | 24. VIS_GetPosition |
| 7. VIS_ReduceNoise | 25. VIS_InitBwimg |
| 8. VIS_ReduceNoisef | 26. VIS_ComputeBW |
| 9. VIS_ReduceNoise2 | 27. VIS_DrawLimits |
| 10. VIS_ReduceNoise2f | 28. VIS_DrawBorder |
| 11. VIS_FindLimits | 29. VIS_MarkObject |
| 12. VIS_FindClass | 30. VIS_ErrorDisplay |
| 13. VIS_FindClasses | 31. VIS_InitShowProgress |
| 14. VIS_MedianHue | 32. VIS_ShowProgress |
| 15. VIS_Init | 33. VIS_ShowOffset |
| 16. VIS_ColClear | 34. VIS_InitCam |
| 17. VIS_ColInit | 35. VIS_AutobrightnessDelay |
| 18. VIS_ColFindOne | 36. VIS_TeamRobot |

**Tabla 6**. Funciones de la librería LibVision

**Nota:**
Podemos utilizar en nuestra aplicación cualquier librería escrita en ANSI-C que utilice memoria estática.