Sistema de Visión

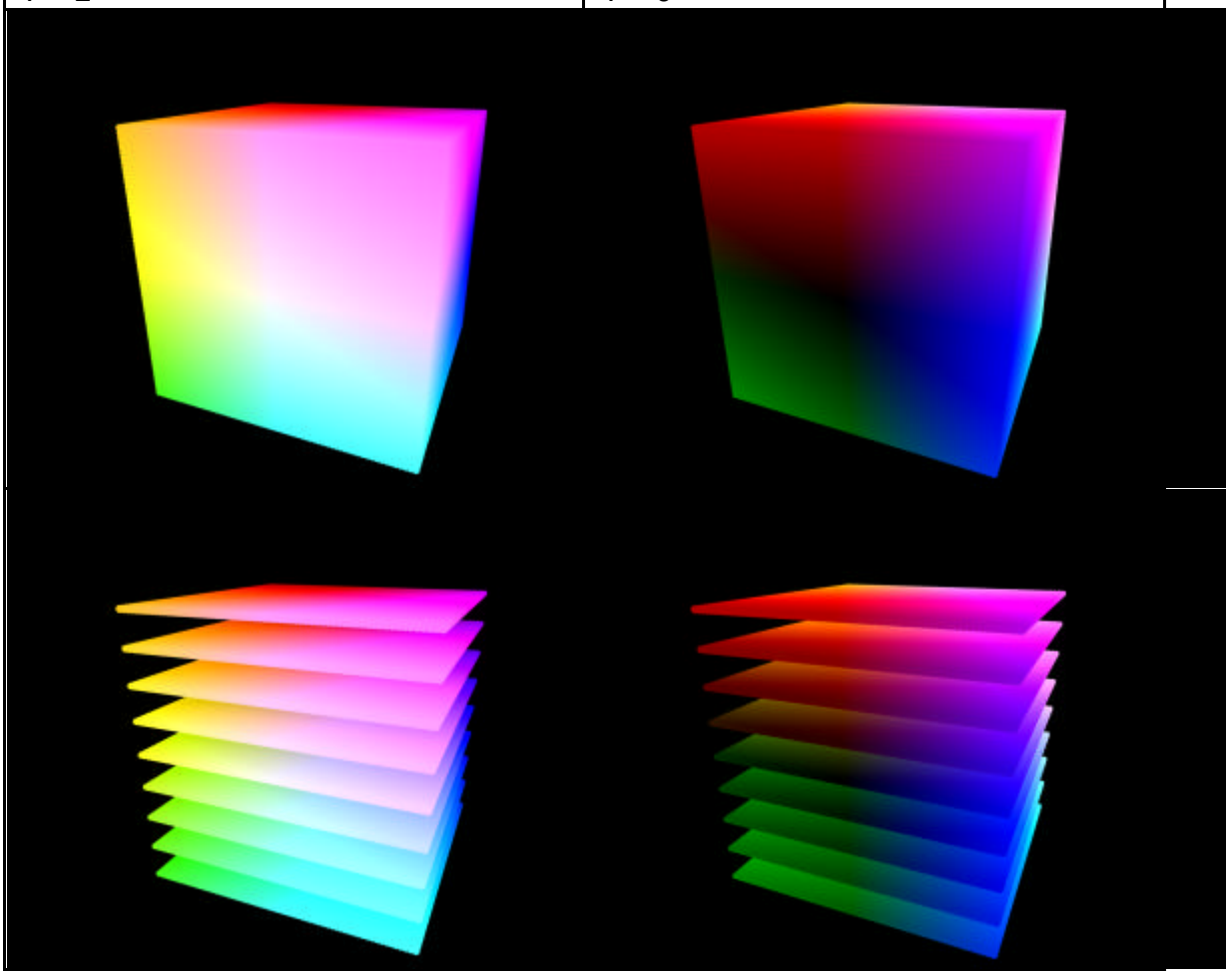
Espacio de color YCbCr

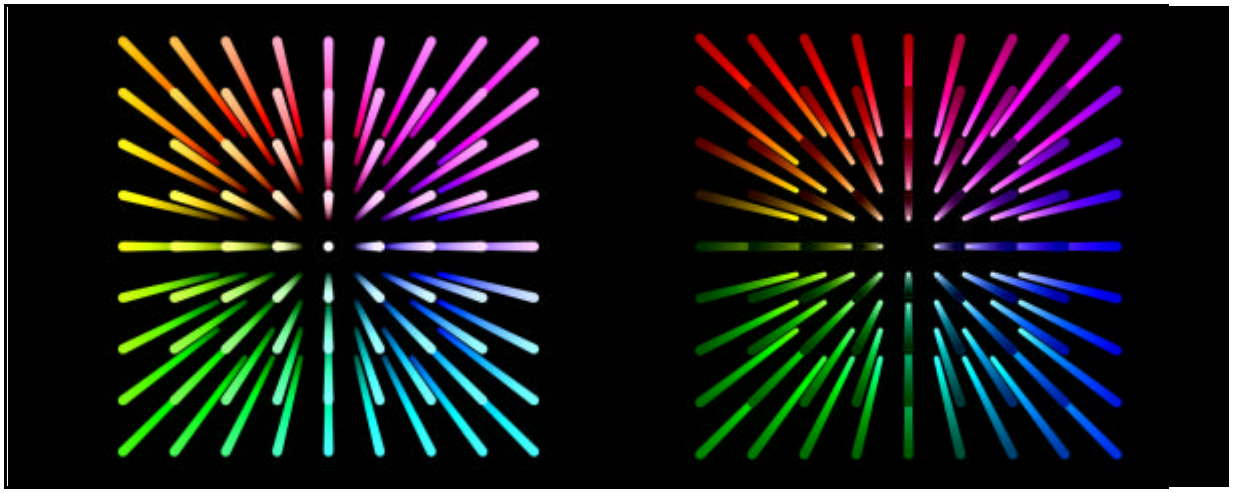
YCbCr es un espacio de color utilizado en sistemas de fotografía y video digital. Define el color en términos de un componente de luminancia y dos de crominancia.

- Y representa la luminancia (luma) y se encuentra en el rango de 0 a 255.
- Cb y Cr representan la crominancia (chroma), los colores azul y rojo respectivamente. Se encuentran en el rango -128 a 127 signado ó 0 a 255 no signado.

Cubo de color YCbCr

| Y = 1 | Y = 0 |
| --- | --- |





Espacio de color RGB

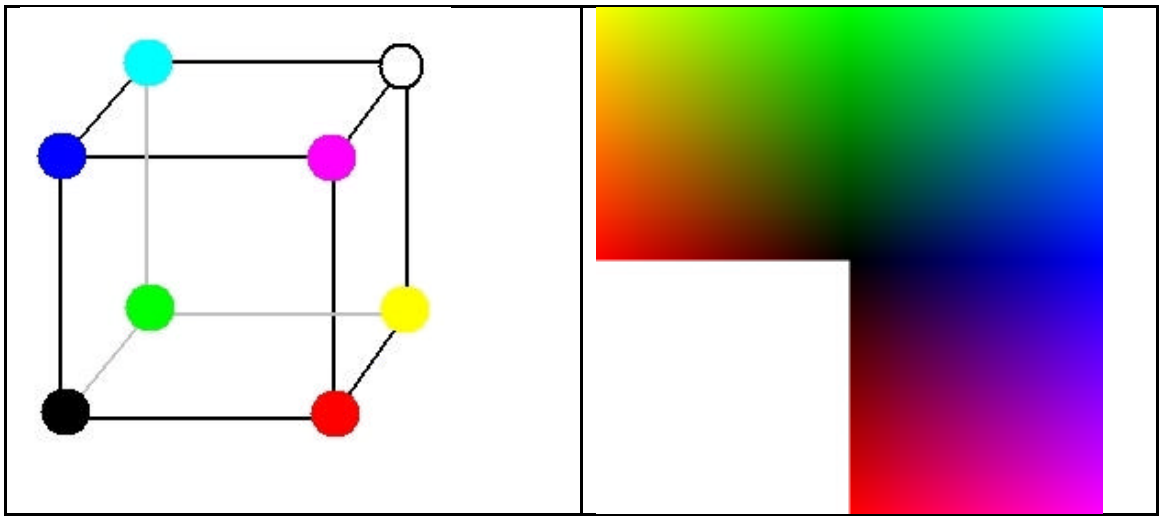
Representa una composición del color en términos de los colores primarios con los que se forma (rojo, verde y azul). Está basado en la síntesis aditiva, es decir la mezcla por adición de los tres colores luz primarios.

La intensidad de cada componente se mide en el rango 0 a 255.

- Rojo: (255,0,0)
- Verde: (0,255)
- Azul: (0,0,255)

El conjunto de todos los colores se representa en un cubo, donde cada color es un punto en la superficie o interior de este cubo. La escala de grises se presenta en la diagonal blanco-negro.

Cubo de color RGB



Convirtiendo de un espacio a otro

YCbCr a partir de RGB

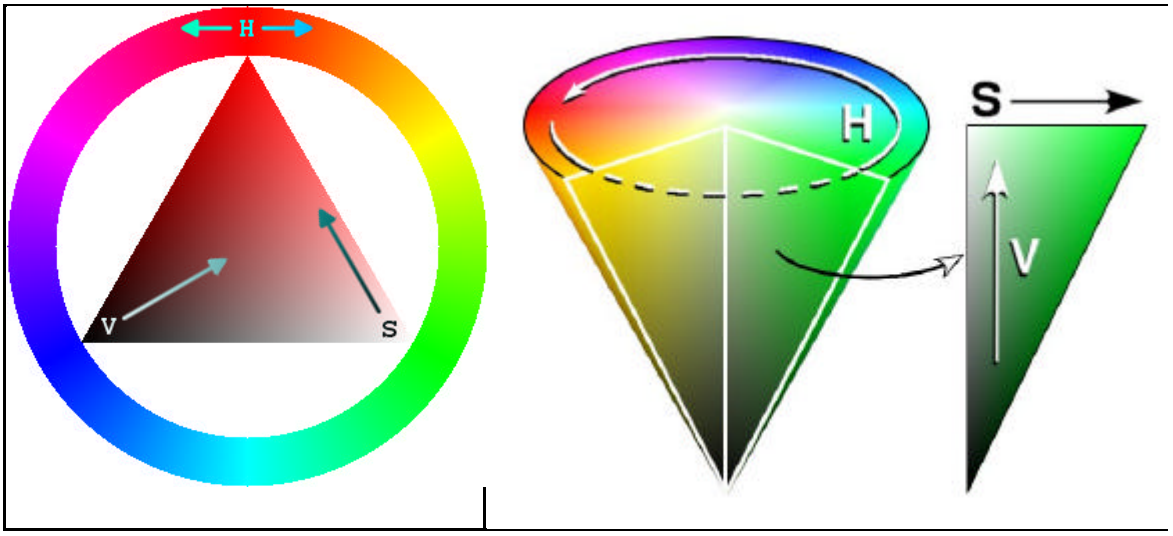
$$Y = 16 + (65.481 \times R + 128.553 \times G + 24.966 \times B)$$
$$Cb = 128 + (-37.797 \times R - 74.203 \times G + 112.0 \times B)$$
$$Cr = 128 + (112.0 \times R - 93.786 \times G + 18.214 \times B)$$

Espacio de color HSV

Define el color en términos de sus componentes en coordenadas cilíndricas.

- Hue/Tonalidad: representa el tipo de color. Es un grado de ángulo de 0 a 360º ó de 0 a 100%. Cada valor corresponde a un color.
  - 0 es rojo.
  - 60 amarillo.
  - 120 es verde.
- Saturation/Saturación: representa la distancia al eje de brillo blanco-negro. Los valores van de 0 a 100%.
- Value/Valor del color: representa el brillo del color, es decir la altura en el eje blanco-negro. Los valores van de 0 a 100%.

Triángulo HSV y cono de color




Convirtiendo de un espacio a otro

HSV a partir de RGB

$$H = \begin{cases} \text{no definido}, & \text{si } MAX = MIN \\ 60° \times \frac{G-B}{MAX-MIN} + 0°, & \text{si } MAX = R \text{ y } G \geq B \\ 60° \times \frac{G-B}{MAX-MIN} + 360°, & \text{si } MAX = R \text{ y } G < B \\ 60° \times \frac{B-R}{MAX-MIN} + 120°, & \text{si } MAX = G \\ 60° \times \frac{R-G}{MAX-MIN} + 240°, & \text{si } MAX = B \end{cases}$$

$$S = \begin{cases} 0, & \text{si } MAX = 0 \\ 1 - \frac{MIN}{MAX}, & \text{en otro caso} \end{cases}$$

$$V = MAX$$

Configurando

Dentro del directorio raíz de la memory stick se encuentra el archivo de configuración Camera.cfg, en él se indican 5 parámetros en la parte superior y debajo de ellos la descripción de qué significa cada uno.

# Camera.cfg

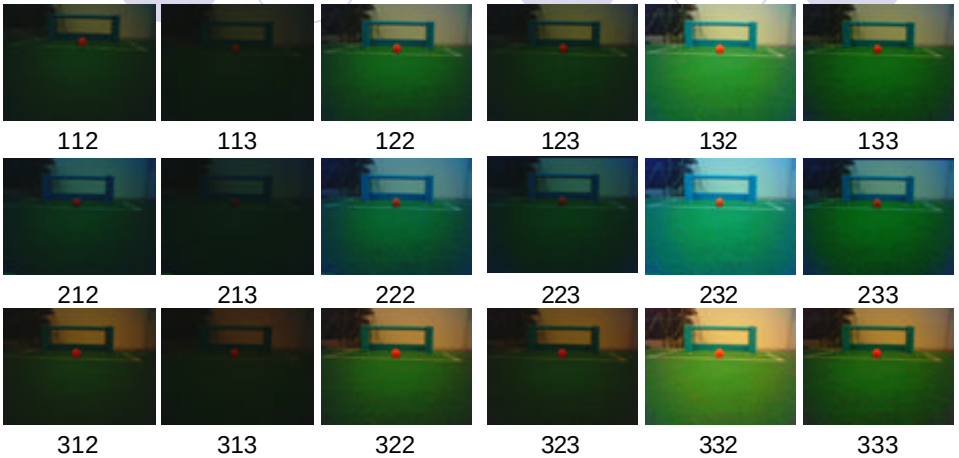
1 3 2 1 2

#File Format:
White_Balance Gain Shutter_Speed Server Photos
#Parameters:
##White Balance:
###WB_INDOOR_MODE = 1
###WB_OUTDOOR_MODE = 2
###WB_FL_MODE = 3
##Gain:
###GAIN_LOW = 1
###GAIN_MID = 2
###GAIN_HIGH = 3
##Shutter Speed:
###SHUTTER_SLOW = 1
###SHUTTER_MID = 2
###SHUTTER_FAST = 3
##Server
###SERVER_OFF = 1
###SERVER_ON = 2
##Photos
###NONE = 0
###ALLCOMBINATIONS = 1
###CAMERACFG = 2
###SEGMENTATION = 3

Los primeros 3 parámetros describen el filtro digital que se utiliza según las condiciones de iluminación. El siguiente parámetro sirve para establecer conexión en caso de que se desee transmitir la imagen. El último indica qué tipo de fotos se tomarán o, en caso de juego, no tomar fotos.

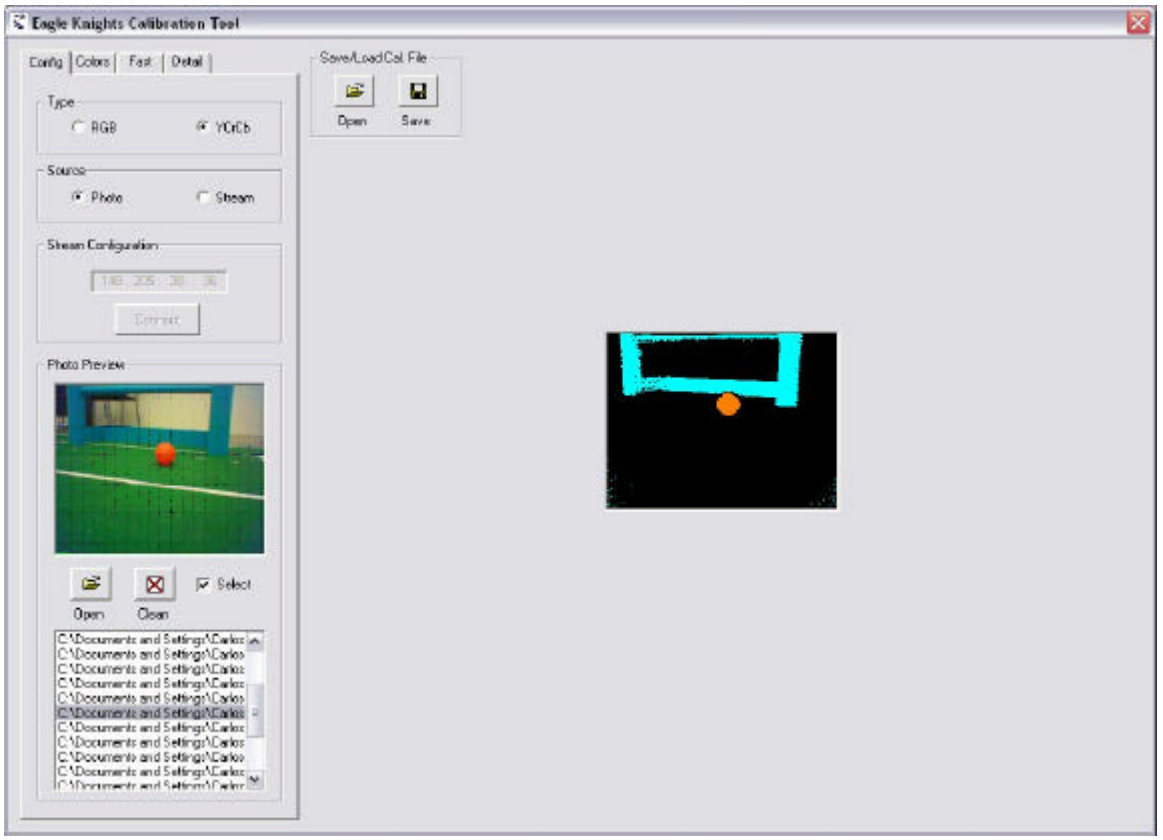
Para la segmentación primero debemos establecer un filtro (los 3 primeros parámetros). Para esto es necesario indicar en el 5to parámetro que queremos tomar fotos con todas las combinaciones (1). Lo que obtenemos es algo como lo siguiente.

# ALLCOMBINATIONS = 1



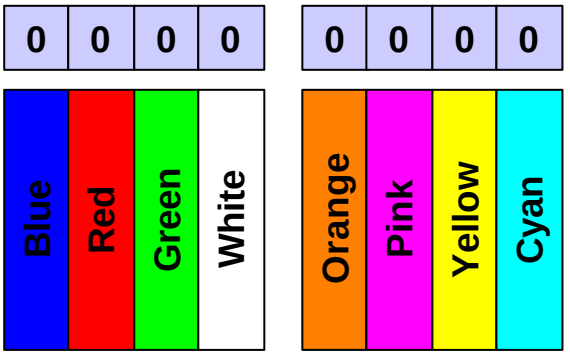
Analizando las fotos se decide qué filtro usar. Lo que sigue es cambiar el parámetro de ‘1’ a ‘2’ y tomar fotos de puntos de interés, como las porterías, postes, pelota y otros robots. Éstas fotos se abren con el programa EKCalibration y se realiza la segmentación.

EKCalibration



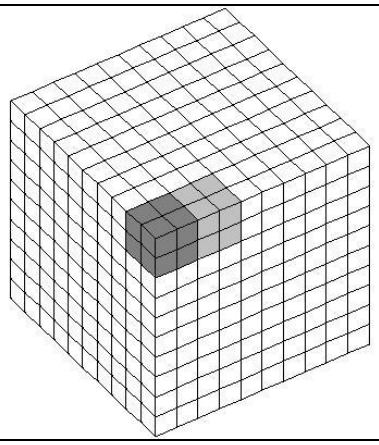

Se utiliza un byte para representar las 8 clases de colores.

## Clasificación del pixel

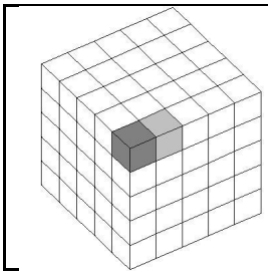


Cubo de Calibración

Ejemplo con espacio de color con 10 niveles para cada componente.

| | |
|---|---|
|  | $10^3$(1Byte) = 1000B |

Ejemplo con 5 niveles.

| | |
|---|---|
|  | $5^3$(1Byte) = 125B |

Con 256 tendríamos $256^3$(1Byte) = 16MB lo cual es demasiado, dado que es la capacidad total de la memory stick que utiliza el Aibo. Entonces utilizamos 128, con lo que se obtiene $128^3$(1Byte) = 2MB.

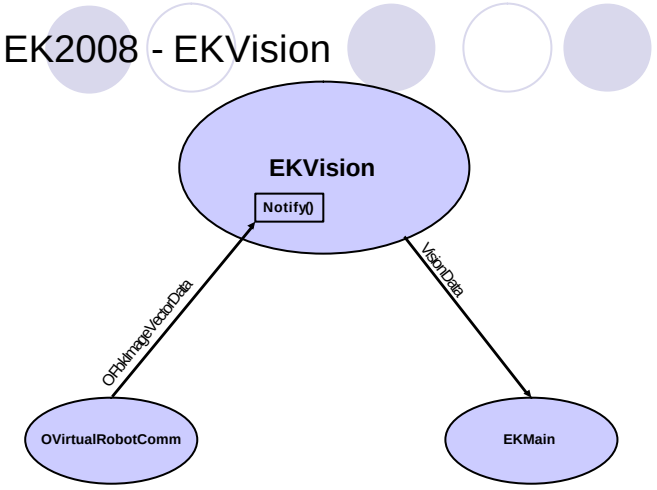
Generación del cubo de calibración

```
void CMainManager::GenerateCube()
{
…
if(imageType == IMAGERGB)
{
…
else
{
int colorProm[8];
for(Y=0;Y<256;Y=Y+2)
for(Cr=0;Cr<256;Cr=Cr+2)
        for(Cb=0;Cb<256;Cb=Cb+2)
        {
                //Clean average
                for(i=0;i<8;i++)
                colorProm[i] = 0;
                for(Y2=Y;Y2<Y+2;Y2++)
                for(Cr2=Cr;Cr2<Cr+2;Cr2++)
                for(Cb2=Cb;Cb2<Cb+2;Cb2++)
                {
                                YCrCb2RGB(Y2,Cr2,Cb2,&Rb,&Gb,&Bb);
                        RGB2HSV(Rb,Gb,Bb,&fH,&fS,&fV);
                        //Color 0
                        if ((fH>= val[0].Hmin && fH<= val[0].Hmax) &&
                        (fS>= val[0].Smin && fS<= val[0].Smax) &&
```

```
                                    (fV>= val[0].Vmin && fV<= val[0].Vmax))
                                    colorProm[0]= colorProm[0]+1;
                                    …
                            }
        Y3 = Y >> 1;
              Cr3 = Cr >> 1;
              Cb3 = Cb >> 1;
                                                    //Color 0
              if (colorProm[0]>1)
                 sYCrCb128[Y3][Cr3][Cb3] |= 0x01;    //Set bit 0
              else
                 sYCrCb128[Y3][Cr3][Cb3] &= 0xFE; //Setbit1
                       ….
                    }
              }
        }
```

## EKVision

Este objeto tiene comunicación con los objetos EKMain, cuando envía datos, y con OVirtualRobotComm, del que recibe la imagen.




Con el método loadEKCalib se carga el cubo de color como un vector.

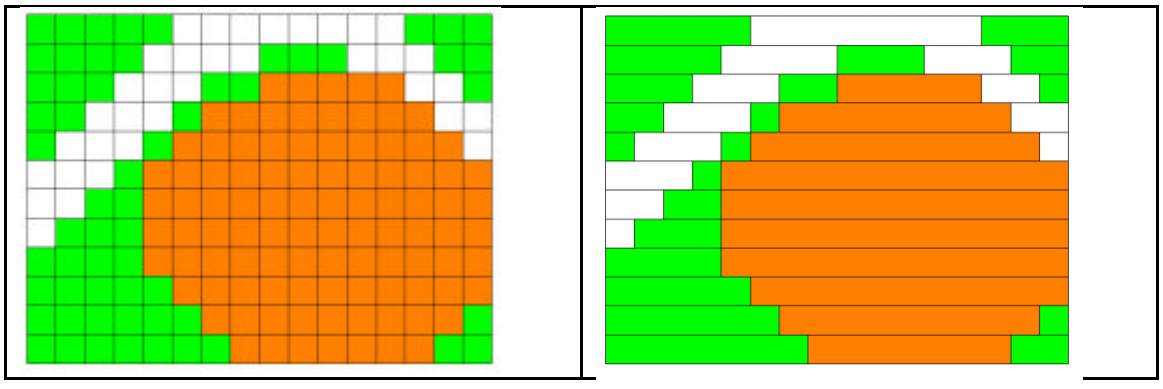
```
void EKVision::loadEKCalib() {

    char *path = "/MS/EKCALIB.ysp";
    byte color;
    int i;

    FILE* fp = fopen(path, "r");
    if (fp == 0) {
       OSYSLOG1((osyslogERROR, "can't open %s", path));
       return;
    }

    for(i=0;i<2097152;i++){
        fread(&color, 1, 1, fp);
        YCrCbCube[i]=color;
    }
    fclose(fp);
}
```

Para obtener la imagen de la cámara se hace lo siguiente.

```
OFbkImageVectorData* fbkImageVectorData       = (OFbkImageVectorData*)event.Data(0);

OFbkImageInfo* info = fbkImageVectorData->GetInfo(ofbkimageLAYER_H);
byte*          data = fbkImageVectorData->GetData(ofbkimageLAYER_H);

OFbkImage yImage(info, data, ofbkimageBAND_Y);
OFbkImage crImage(info, data, ofbkimageBAND_Cr);
OFbkImage cbImage(info, data, ofbkimageBAND_Cb);

for (int y = infoheader.height - 1 ; y >= 0; y --) {
   for (int x = 0; x < infoheader.width; x++) {
       Y = yImage.Pixel(x, y) >> 1;
       Cr= crImage.Pixel(x, y)>> 1;
       Cb= cbImage.Pixel(x, y)>> 1;
       cubito=YCrCbCube[128*128*Y+128*Cr+Cb];
       imagen[x][y]=cubito;
   }
}
```

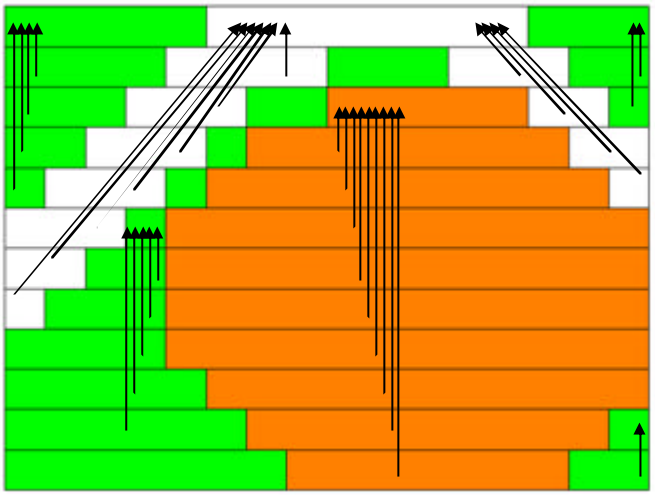
Para convertir de pixeles a renglones se utiliza la función EncodeRuns, que regresa el número de regiones que se forman.

```
int EKVision::EncodeRuns() {
   int cntX=0;
   int cntRLE=0;
   int RLEcolor;
   int len;
   int j;
   for(j=0;j<DEFAULT_HEIGHT;j++) {
       cntX=0;
      while(cntX<DEFAULT_WIDTH) {
       RLEcolor=imagen[cntX][j];
       len=0;
      while(cntX<DEFAULT_WIDTH &&  imagen[cntX][j]==RLEcolor)) {
             len++;
             cntX++;
       }
       rmap[cntRLE].color = RLEcolor;
       rmap[cntRLE].length = len;
       rmap[cntRLE].parent = cntRLE;
       cntRLE++;
       }
   }
   return cntRLE--;
}
```



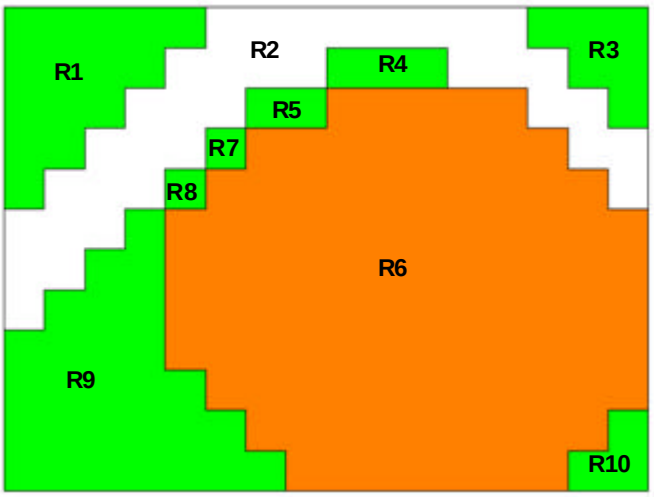
Después con el método ConnectComponents los blobs apuntan a su “padre”, que es el que colinda con ellos en la parte superior.

## ConnectComponents



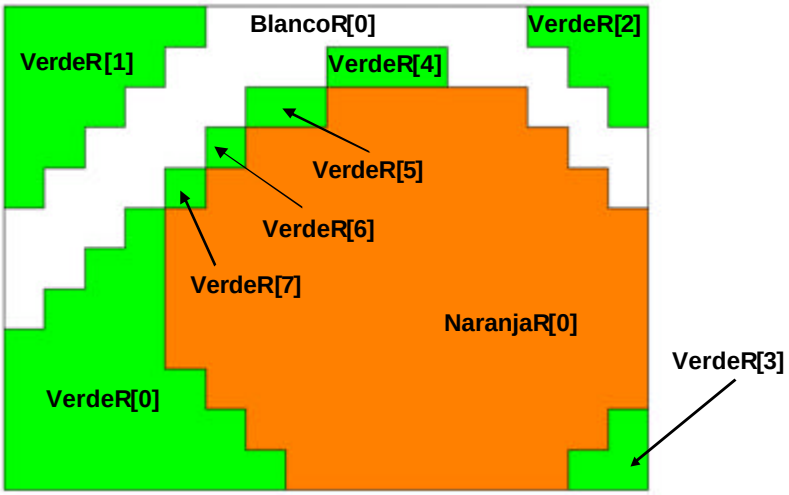
Después de esto se extraen las regiones con el método ExtractRegions, pero las regiones resultantes no se encuentran ordenadas.

## ExtractRegions




Luego se ordenan según el área, la más grande es la primera en indexarse. Esto se hace con el método SeparateRegions.

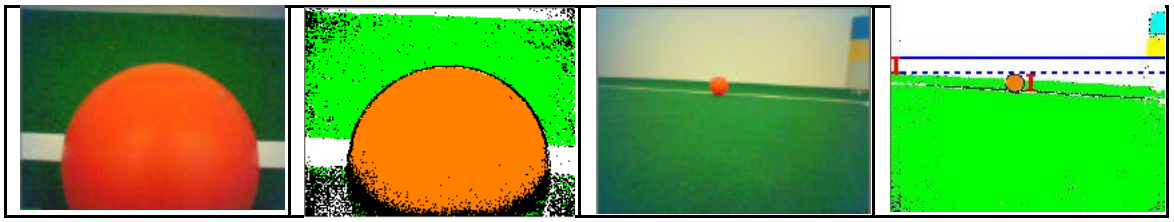
# Separate Regions




Después del procesamiento que se da a la imagen se reconocen los objetos. Para el reconocimiento de la pelota se utiliza un discriminante según el área de color naranja que se observa.

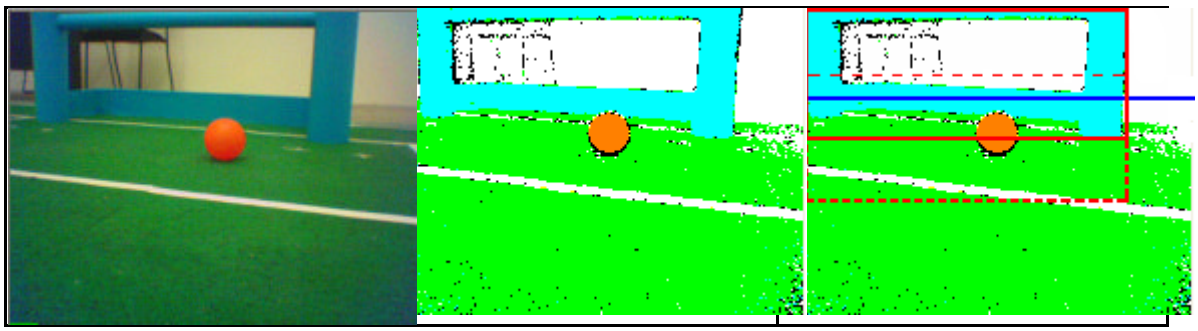
Si el área es mayor a 2000 pixeles se reconoce como pelota, pero si el área apenas es mayor a 5 pixeles deben evaluarse entonces los siguientes puntos:

- Encontrar la coordenada en Y que limita superiormente el área de la cancha (coorY).
- Calcular la altura de la región candidata a ser pelota.
- El centroide de la región candidata debe estar debajo de un límite teórico definido por el límite de la cancha y la altura de la pelota.



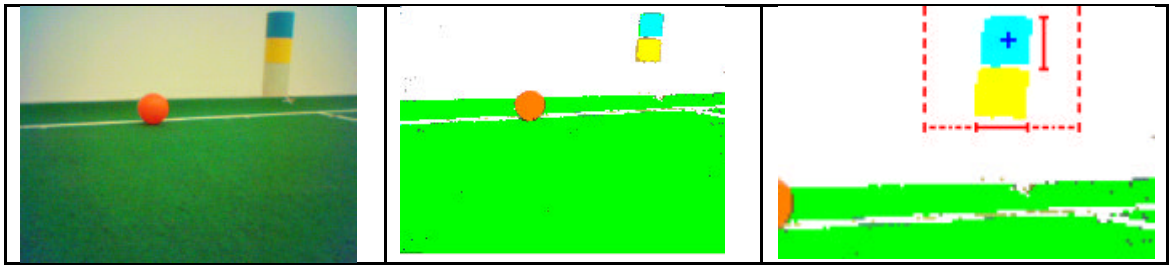
Para el reconocimiento de las porterías se siguen los siguientes puntos.

- Determinar la altura de la región candidata.
- Sumar a la coordenada Y2 de la región candidata la mitad de la altura para obtener Yaux.
- El límite superior de la cancha debe estar por encima de Yaux.



Para el reconocimiento de postes se realiza lo siguiente.

- Se calcula largo y ancho de la región amarilla candidata, el mayor (llamado lado).
- Se evalúan todas las regiones cyan:
  - El centroide en X de la región cyan está comprendida en un ancho de 3 veces el lado, centrado en centroide X de la amarilla.
  - El centroide en Y debe estar dentro de un alto comprendido entre:
    - Si es poste Cyan-Amarillo: el límite superior en Y de la región amarilla y el valor de lado desde ese límite hacia arriba.
    - Si es poste Amarillo-Cyan: el límite inferior en Y de la región amarilla y el valor de lado desde ese límite hacia abajo.



La estructura EKVisionData contiene un booleano para indicar si el objeto está siendo observado, los valores que lo limitan en X y Y y las coordenadas del centroide, así como el área medida en número de pixeles.

```
struct Objeto {
        bool    visto;
        int     x1;
        int     y1;
        int     x2;
        int     y2;
        float   cen_x;
        float   cen_y;
        int     area;
        …
};
```