Reconocimiento de imágenes en fútbol de robots
Informe de trabajo final para Tiempo Real
Licenciatura en informática - Plan 90

Ignacio Jaureguiberry
UNLP
3319/7

18 de septiembre de 2011

Índice

Índice de figuras

1. Introducción

En este informe se detallan los resultados de un trabajo final para la materia *Diseño de sistemas de tiempo real*.

El trabajo consiste en un sistema de visión por computador para el reconocimiento de objetos en un partido de *Fútbol de robots*.

El fútbol de robots se puede definir como una competición de tecnología robótica de avanzada en un espacio contenido [15].

> Desde el principio, el fútbol de robots ha sido una plataforma para la investigación y desarrollo de robots móbiles e independientes y sistemas multi-agente, involucrando las áreas más diversas de ingeniería y ciencias de computación [6].

Las competencias internacionales de fútbol de robots más reconocidas son las de *Federation of International Robot-soccer Association (FIRA)* [15] y *RoboCup* [16]:

> Las competiciones RoboCup proveen un excelente canal para la diseminación y validación de conceptos y enfoques innovadores para robots autónomos bajo condiciones muy desafiantes y adversas.

Este trabajo se acota a una categoría particular de la competencia propuesta por la *FIRA*: la liga mediana. En esta liga, los equipos se componen de 5 robots controlados por un sistema centralizado. Este sistema compuesto por una única computadora se vale de la imagen de una única cámara situada sobre el campo de juego.

Una forma de definir el sistema de control es dividir el problema en las siguientes áreas:

- Reconocimiento del campo: usando la imagen de la cámara, y posiblemente información anterior, se determina la posición, velocidad y orientación de los robots de ambos equipos y de la pelota.
- Planificación de las acciones de los robots: Se determina las acciones a tomar con objeto de lograr el objetivo de trasladar la pelota al objetivo.
- Control de los robots: Se usa un sistema de comunicación inalámbrico para mover los robots de acuerdo a la estrategia definida.

Este trabajo se enfoca en el primero de los desafíos, el reconocimiento de campo: procesar la imagen de la cámara para determinar la posición de los robots y la pelota, haciéndolo en el mínimo tiempo posible.

1.1. Reglas de MiroSot

Las reglas de fútbol de robot usadas son las de la categoría *MiroSot* que se pueden consultar con más detalle en [15].

Un resumen de las mismas es:

Micro Robot World Cup Soccer Tournament

Un partido será jugado por dos equipos, cada uno consistente de cinco robots, donde uno de ellos puede ser el *goalkeeper*. A tres miembros humanos en el equipo, el *manager*, el *coach* y el *trainer*, se les permite permanecer en el campo.

Se puede usar solo una computadora por equipo, principalmente dedicada a procesamiento de visión y otras identificaciones de ubicación.

El tamaño de cada robot será limitado a 7,5 cm x 7,5 cm x 7,5 cm. El alto de la antena no se considerará para decidir el alto del robot.

Los miembros humanos no pueden participar del juego a menos que el mismo esté en pausa.

- Robot: tiene que ser de a lo sumo 7,5 cm x 7,5 cm x 7,5 cm.
- Pelota: se usa una pelota de golf naranja.
- Campo de juego: 400 *cm* x 280 *cm* para la liga grande, y 220 *cm* x 180 *cm* para la liga mediana.

1.2. Tiempo real

El sistema construido no funciona en tiempo real "duro", pero si provee de herramientas para compensar las latencias en la detección.

El sistema de estrategia que define las acciones de los robots en la competencia tiene que considerar las acciones basados en las diferencias entre lo sensado y el estado en el momento que el robot recibe la orden. Por ejemplo, considerando solo la pelota, si la misma está en movimiento, la ubicación en que la misma se encuentre cuando el robot reciba un comando es distinta a la ubicación de la pelota cuando se tomo la imagen de donde se extrae su ubicación. Hay varios procesos que toman tiempo:

- Latencia de la cámara, desde que toma la imagen hasta que la entrega al sistema de detección.
- Latencia de la detección.
- Latencia de definición de la estrategia.
- Latencia de comunicación.

Este trabajo se enfoca en la detección, y la latencia de procesamiento, desde que el sistema recibe la imagen hasta que termina el sensado, se devuelve al programa cliente. Se pueden conocer 2 datos de importancia:

- Cuanto tiempo tomó el procesamiento en nanosegundos[1].
- El momento en el que la imagen entró al sistema de visión, denotando cuando, en el pasado, la imagen era válida.

2. Ambiente de pruebas

Para el desarrollo del sistema, se construyó una maqueta de pruebas compuesta de los siguientes elementos:

- Un conjunto de "robots" de prueba, constituido por 10 robots de cartón, 5 por equipo, con patrones de color en las "camisetas".

[1] La precisión de la medida de tiempo en nanosegundos están limitados a las capacidades de los relojes del sistema.

- Un campo de juego, constituido por un fondo negro y algunos artefactos de iluminación. El fondo se creo usando cartulina mate color negro.
- Un dispositivo de captura: cámara de video *Sony Handycam DCR-DVD108 NTSC* que captura videos en soporte DVD, con resolución $720x480$ pixeles en 24 bits, a 29,970 frames por segundo.

En la Figura 1 se pueden ver dos imágenes mostrando el campo de juego, y la ubicación de la cámara.

Figura 1: "Campo de juego"

2.1. Robots

Según las reglas de juego, los robots están limitados a un espacio de 7,5 cm x 7,5 cm x 7,5 cm, sin incluir la antena. La pelota es una pelota de golf color naranja de 42,7 mm de diámetro. En la liga mediana, el campo de juego es de 220 cm x 180 cm.

Respecto de las "camisetas" de los robots, las reglas dicen:

- Ningún robot puede tener el color naranja en su "camiseta".
- El color del equipo es celeste o amarillo, designado por los organizadores. Todo robot tiene que tener una región de al menos 3,5 cm x 3,5 cm con el color de su equipo visible en el tope.
- La "camiseta" de un robot no puede tener ninguna zona con el color del equipo oponente.

Para el ambiente de pruebas, se construyeron maquetas de robots y de la pelota a escala: robots cuadrados de 3 cm de lado con una pelota de 17 mm de diámetro usando cartón. Si bien la altura de los robots es inapropiada, como la cámara los captura desde arriba no tendría que haber diferencia.

Como los robots de 7,5 cm de lado se construyeron de 3,0 cm de lado, las otras medidas se escalan a los siguientes tamaños:

- Campo de juego original de 220 cm x 180 cm pasa a 88 cm x 72 cm.
- Pelota de 42,7 mm de diámetro pasa a 17,08 mm de diámetro.

Las "camisetas" de los robots para este trabajo se segmentan en 4 zonas cuadradas de color como se muestra en la Figura 2.

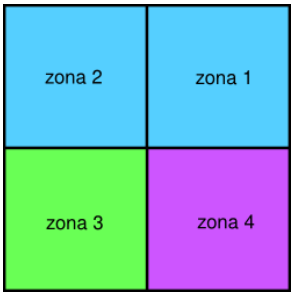

Figura 2: Segmentación de la "camiseta" de un robot en zonas

Para identificar a cada uno de los robots de cada equipo se usa el siguiente sistema de colores: dado el color principal del equipo (amarillo o celeste), los robots se identifican entre si con una combinación del color principal, el color verde y el color violeta.

Los patrones de colores son, llamando KEY al color del equipo:

- robot 0: KEY, KEY, verde, verde.
- robot 1: KEY, KEY, violeta, violeta.
- robot 2: KEY, KEY, verde, violeta.
- robot 3: KEY, KEY, violeta, verde.
- robot 4: KEY, violeta, KEY, verde.

En la Figura 3 se muestran los robots del equipo celeste, ordenados desde el robot 0 (izquierda) al 4 (derecha). El equipo amarillo es igual sustituyendo el color celeste por amarillo.

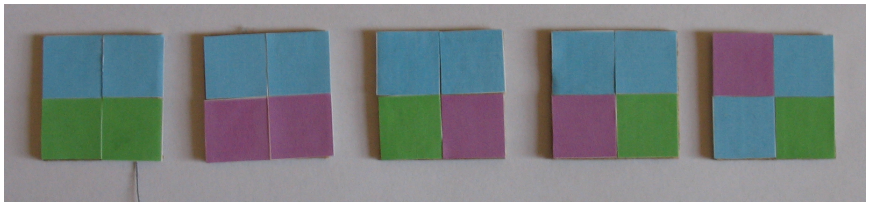

Figura 3: Robots del equipo celeste

Los colores usados para los robots se muestran en el Cuadro 1

Color	Azul	Verde	Rojo
Amarillo	84	245	255
Celeste	255	204	85
Verde	86	255	106
Violeta	255	85	205

Cuadro 1: Colores usados para los robots

3. La librería de reconocimiento *BotTracker*

El trabajo constituye una librería (API) para procesar imágenes en un partido de fútbol de robots: la librería *BotTracker*.

Los principales objetivos de esta herramienta son:

- Proveer una interfaz que sintetice la información en video a información utilizable por otros componentes, con un mínimo de complejidad.
- Que el sistema funcione con tiempos de respuesta razonables para la tarea.
- Proveer al usuario de la interfaz con información de tiempo de respuesta para poder continuar el trabajo compensando la demora en la detección.
- Que el sistema sea robusto, con uso razonable de recursos.

3.1. Interfaz de programación

La librería es orientada a objetos y tiene como principal componente la clase `bot_tracker`. Se construye una instancia de la misma que se usa para procesar los frames del video a medida que se decodifican. En la Figura 4 se puede ver un diagrama de clases de la misma.

La instancia de `bot_tracker` se inicializa con una imagen de fondo y un tamaño de historial a mantener. La imagen de fondo es la misma imagen de la cámara donde no hay ningún robot o pelota, solo el fondo. El tamaño del historial define cuantas instancias de capturas pasadas se mantienen en memoria:

```
cv::Mat background = cv::imread("fondo.bmp");
bot_tracker bt (background, 1);
```

La instancia de `bot_tracker` necesita ser configurada con los parámetros del ambiente de juego. Es necesario establecer la muestra de color para cada uno de los colores naranja de la pelota, amarillo y celeste de los colores de equipo, y el verde y violeta de identificación:

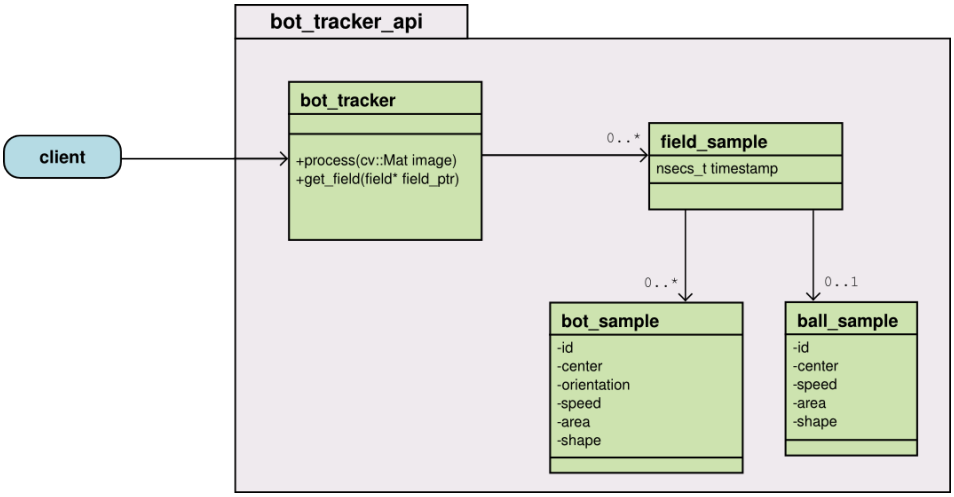

Figura 4: Diagrama de clases de la librería *BotTracker*

```
bt.set_color_code(YELLOW   , 0.33, 0.82, 1.00);
bt.set_color_code(LIGHTBLUE, 1.00, 0.82, 0.61);
bt.set_color_code(VIOLET   , 0.91, 0.66, 1.00);
bt.set_color_code(GREEN    , 0.55, 1.00, 0.75);
bt.set_color_code(RED      , 0.32, 0.43, 1.00);
```

También hay que establecer el nivel de *umbral* que sirve para la separación de los robots del fondo del video como se explica en 5.1:

```
bt.set_threashold(25);
```

Se establece el nivel de *umbral* para aceptar una aproximación del color naranja de la pelota:

```
bt.set_redcolor_threashold(0.65);
```

Y se establecen los límites mínimo y máximo de áreas para la pelota y los robots:

```
bt.set_bot_area_limits(600, 1200);
bt.set_ball_area_limits(100, 200);
```

Una vez inicializada la instancia de `bot_tracker`, se va alimentando la misma con los frames de video, usando el método `process`:

```
capture >> frame;
while (frame.data) {
        nsecs_t dt = bt.process(frame);
        // procesar informacion encontrada.
        capture >> frame;
}
```

La ejecución de `process` devuelve el tiempo que tomo el procesamiento de la imagen en nanosegundos. Además, `bot_tracker` agregar una instancia de `field_sample` a su historial que contiene los datos de robots y pelota encontrados, junto con un *timestamp* que denota el instante en que dicha información era válida. La instancia de `field_sample` se puede obtener con el método `get_field`:

```
field_sample * fs;
bt.get_field(fs);
```

En el Listado 1 se muestra un ejemplo de programa que usa *BotTracker*.

Listado 1 Ejemplo de uso de *BotTracker*

```
#include "bottracker.hpp"
#include <opencv2/opencv.hpp>
#include <opencv2/highgui/highgui.hpp>
int main ()
{
        cv::Mat background = cv::imread("fondo.bmp");
        cv::Mat frame;
        cv::VideoCapture capture("juego.avi");

        bot_tracker bt (background, 1);
        bt.set_color_code(YELLOW, 0.33, 0.82, 1.00);
        // establecer los demas colores...
        bt.set_threashold(25);
        bt.set_redcolor_threshold(0.65);
        bt.set_bot_area_limits(600, 1200);
        bt.set_ball_area_limits(100, 200);

        capture >> frame;
        while (frame.data) {
                bt.process(frame);
                // procesar informacion encontrada.
                capture >> frame;
        }
        return 0;
}
```

4. El programa “cliente”

Se acompaña la librería desarrollada con un programa cliente. El programa usa la librería para el procesamiento, configurando los parámetros para algunos videos de prueba. Este programa abre un archivo de video que se pasa por línea de comandos, y envía los frames a la librería de reconocimiento. Para cada frame procesado, usa los datos devueltos para mostrar en una imagen el frame y los robots encontrados. En la Figura 5 se puede ver la salida de dicho programa.

En esta imagen se puede ver el frame donde se muestran los datos de los robots (centro, velocidad, orientación, área) y de la pelota (centro, velocidad, área). En una línea abajo de la imagen se muestran datos de tiempos de procesamiento, entre ellos, el tiempo de procesamiento del frame, el tiempo promedio de procesamiento, el desvío estandar, el tiempo mínimo y el tiempo máximo. Además, cada robot está rodeado por el contorno donde

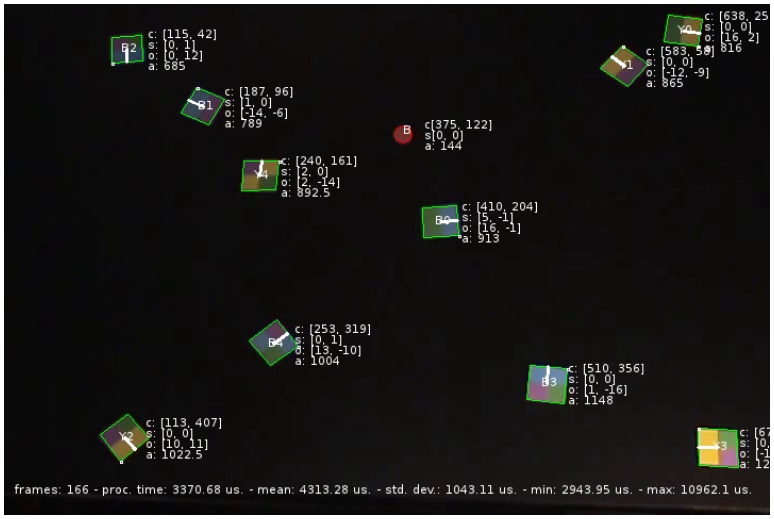

Figura 5: Captura de la interfaz del programa que usa *BotTracker*

se lo ubicó con una línea blanca mostrando su orientación, mientras que la pelota está rodeada por su círculo.

Para iniciar el programa se ejecuta el siguiente comando:

```
$ ./tracktest <archivo con el video de la captura>
```

El programa provee interacción usando el teclado:

- La tecla *espacio* pausa/continúa la animación del reconocimiento.
- La tecla *escape* sale del programa.
- La tecla *s* muestra/oculta las estadísticas.
- La tecla *p* muestra/oculta los perímetros de robots y pelota.
- Las teclas + y - aumentan y disminuyen la velocidad del video respectivamente.
- La tecla . sirve para entrar en modo cuadro por cuadro, donde cada vez que se la presiona se muestra el cuadro siguiente.

5. Procesamiento

El procesamiento de la imagen realiza los siguientes pasos:

1. Detección de blobs. Se analiza la imagen para separar el fondo de las características que podrían ser potenciales robots o la pelota.

2. Vectorización de blobs en contornos. Cada blob se vectoriza en polígonos que definen su forma aproximada.

3. Detección de robots y pelota en contornos. Cada polígono vectorizado se analiza para detectar si son robots o pelota, revisando su geometría y los colores en su interior.

4. Construcción de la muestra. Se sintetiza la información obtenida en un objeto *field* que resume la información obtenida.

5.1. Detección de Blobs

La imagen de frame actual se procesa para obtener una imagen en blanco y negro, donde blanco identifica una sección de la imagen que cambio respecto del fondo y negro identifica sección de la imagen que no cambio.

La detección involucra los siguientes pasos:

1. Se convierte el frame a una imagen en escala de grises.

```
cv::cvtColor(frame, tmp0, CV_BGR2GRAY);
```

2. Se calcula una imagen nueva mediante una diferencia absoluta con la imagen de fondo, previamente convertida a escala de grises.

```
cv::absdiff(tmp0, background_gray, tmp1);
```

3. Se umbraliza [2] la imagen a blanco y negro: todo pixel de la diferencia absoluta que tenga una valor mayor al nivel de umbral (threshold) se considera blanco, mientras que el resto se consideran negros.

```
cv::threshold(tmp1, tmp2, threashold, 255, CV_THRESH_BINARY);
```

La Figura 6 muestra la imagen original en escala de grises, la imagen resultando de la diferencia absoluta con el fondo y la imagen umbralizada.

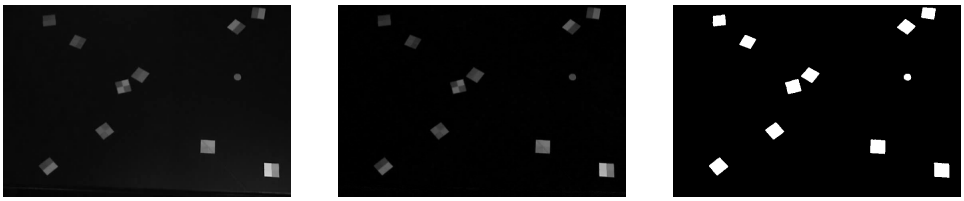

Figura 6: Umbralización de una captura con el fondo

Naturalmente, `threshold` es uno de los parámetros con los que se inicializa `bot_tracker`, ya que este valor depende de el ambiente.

[2] En este texto, se usa un verbo *umbralizar* para denotar denotar la operación de separación de características usando un valor umbral, aunque este verbo podría no existir.

5.2. Vectorización

En esta parte del procesamiento, la imagen umbralizada en blanco y negro se transforma en polígonos:

```
vector< vector<Point> > contours;
vector< vector<Point> >::iterator c_it;
vector<Point> polygon;

findContours(tmp2, contours, CV_RETR_EXTERNAL, CV_CHAIN_APPROX_NONE);
for (c_it = contours.begin(); c_it != contours.end(); ++c_it)
{
        approxPolyDP(cv::Mat(*contour_it), polygon, 5, true);
        // procesar polígono.
}
```

En `findContours` se transforma la imagen blanco y negro a un conjunto de contornos. Cada contorno es un conjunto de puntos que forman el contorno de una de las regiones de la imagen con pixeles blancos contiguos. `findContours` está basado en el algoritmo de [1]. Para esta implementación se usa detección de bordes externos solamente.

Para cada uno de los contornos en el conjunto, se simplifica el polígono usando `approxPolyDP`, que reduce la cantidad de puntos del contorno.

`approxPolyDP` devuelve estas líneas como la lista de sus puntos extremos, en sentido anti-horario. Está basado en el algoritmo de Douglas–Peucker, detallado en [2].

En la Figura 7 se muestran los pasos aplicados a un robot: a la izquierda está la imagen blanco y negro, al medio el resultado de `findContours`, y a la derecha el resultado de `approxPolyDP` que produce un polígono de 4 lados.

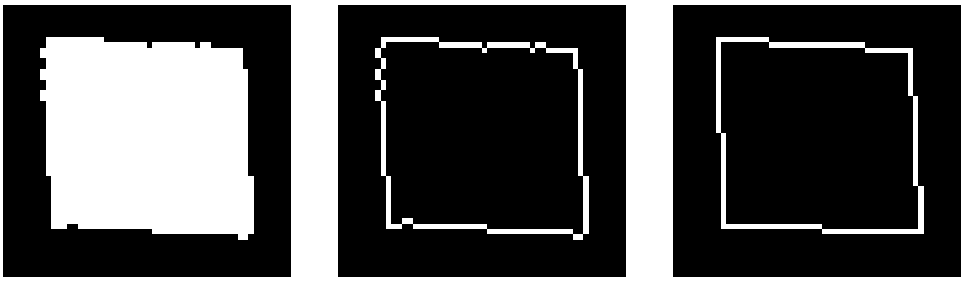

Figura 7: Vectorizado de una imagen binaria

5.3. Enumeración de robots

Ya obtenidos los polígonos de los blos, se intenta ubicar cuales de los mismos son robots y cual es la pelota.

El proceso se hace en 2 intentos. En un primer intento se intenta determinar si el polígono es un robot o la pelota basándonos en su área y la cantidad de sus lados. Si el primer intento falla, el polígono se agrega a una lista de polígonos que van a ser procesados por el segundo método, más complejo computacionalmente.

El Listado 2 muestra un pseudo-código de esta operación.

Listado 2 Primer intento de detección de robot o pelota

```
if (polygon.size() == 4 && isContourConvex(polygon)) {
        robot_id = try_robot(polygon);
        if (robot_id != ROBOT_UNKNOWN) {
                robot_found = true;
        }
        if (!robot_found) {
                ball_found = try_ball(polygon);
        }
}
if (!robot_found && !ball_found)
        unidentified.push(polygon);
```

5.3.1. Detección de un robot

El algoritmo para detectar robots requiere de un polígono de 4 lados, definido por los 4 puntos de sus líneas.

El primer paso es asegurar que el área del polígono se corresponde con la de un robot, usando los límites de área mínima y máxima con que se inicializó *BotTracker*.

Si esa verificación no falla, se extrae la información de color de cada subsección del robot, evaluando el color en el centro de cada cuadrado de color.

Para el cálculo de cada centro, se usan las diagonales del cuadrado, donde cada diagonal tiene el centro de 2 de las subsecciones.

Usando la Figura 8 como referencia: en 0,25 de la diagonal $diag_1$ está el centro de la subsección s_1 que llamamos c_1, y en 0,75 el de la subsección s_3, que llamamos c_3.

La diagonal $diag_1$ es el vector $p_2 - p_0$. El centro de la sección s_1 está en c_1:

$$c_1 = 0{,}25 * diag_1 + p_0 \tag{1}$$

El centro de la sección s_3 está en c_3:

$$c_3 = 0{,}75 * diag_1 + p_0 \tag{2}$$

De igual forma se calculan los centros c_2 y c_4 usando la diagonal $diag_2 = p_3 - p_1$.

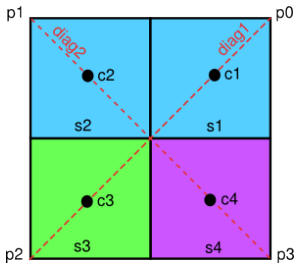

Figura 8: Detección de los colores de un robot

Una vez encontrado en centro de una subsección, se calcula el color promedio de la misma en una región de $5x5$ alrededor del centro, usando `cv::mean` y entonces se prosigue con la identificación del robot.

5.3.2. Identificación de un robot

Teniendo los colores c_1, c_2, c_3 y c_4 de las secciones de un robot, identificamos el mismo de la siguiente forma:

1. Verificamos el color de la "camiseta" del robot: celeste o amarillo. Para esto, tenemos que ver si alguno de los 4 colores $c_1 \cdots c_4$ es amarillo. Como no se permite en la competencia que un robot del equipo celeste tenga color amarillo en su "camiseta", y que un robot del equipo amarillo tenga color celeste, se puede garantizar a que equipo pertenece el robot sin ambigüedad.

   El color encontrado de "camiseta" se denomina KEY.

2. Se verifica si $c_1 = c_3 = KEY$, o si $c_2 = c_4 = KEY$, o sea, si los colores del equipo están en secciones opuestas. Si es así, teniendo solo 1 de los patrones de colores con esa configuración (es el último robot en la Figura 3), determinamos la identidad de dicho robot: Robot 4.

3. Si los colores de la camiseta no están en secciones opuestas, tenemos que rotar los colores hasta que $c_1 = c_2 = KEY$, con el siguiente algoritmo:

```
void rotate (color &c1, color &c2, color &c3, color &c4) {
        color &temp = c1;
        c1 = c2; c2 = c3; c3 = c4; c4 = temp;
}

while (c1 != KEY and c2 != KEY) {
        rotate (c1, c2, c3, c4);
}
```

   Entonces, con los colores ordenados podemos determinar la identidad del robot mirando los colores c_3 y c_4, los casos son:

- (Verde, Verde): Robot 1.
- (Violeta, Violeta): Robot 2.
- (Verde, Violeta): Robot 3.
- (Violeta, Verde): Robot 4.

5.3.3. Detección de la pelota

La detección de la pelota es más sencilla que los robots, ya que solo tiene un color. Siempre que el área se corresponda con el área de pelota esperado, se calcula el color promedio en una región de $3x3$ alrededor de su centro y se verifica que dicho color sea compatible con el naranja establecido como patrón en la inicialización de *BotTracker*.

5.3.4. Comparación de color

Dadas las variaciones de color que son producto de las diferencias de iluminación, o de las características de la cámara entre otros factores, la detección de un color se tiene que hacer por aproximación.

El mecanismo usado es el error cuadrado entre los valores BGR (Blue-Green-Red) encontrados y los que se establecen como patrón al inicializar *BotTracker*.

Para un color dado, el amarillo por ejemplo, se calcula la diferencia de color Δy como:

$$\Delta y = (c_0 - Y_0)^2 + (c_1 - Y_1)^2 + (c_1 - Y_1)^2 \tag{3}$$

Donde c_0, c_1, c_2 son las componentes azul, verde y roja respectivamente de la muestra obtenida, y Y_0, Y_1, Y_2 son las componentes azul, verde y roja respectivamente del color patrón establecido en la inicialización de *BotTracker*.

En el caso de los robots, de las diferencias de color con el amarillo (Δy), celeste (Δl), verde (Δg) y violeta (Δv), nos quedamos con el de menor valor. Este es el que define si el color obtenido es amarillo, celeste, verde o violeta respectivamente.

En el caso del color de la pelota, hay que corroborar el color contra el naranja establecido como patrón (O_0, O_1, O_2).

Para eso se calcula el valor σ como:

$$\sigma = 1 - \sqrt{\Delta o} = 1 - \sqrt{(c_0 - O_0)^2 + (c_1 - O_1)^2 + (c_2 - O_2)^2} \tag{4}$$

Este valor σ lo denominamos "certeza" de la verificación. Notar que mientras más parecidos sean los valores de la muestra (c_0, c_1, c_2) a los del color patrón (O_0, O_1, O_2), σ será más próximo a 1. Decimos entonces que el color de la muestra se corresponde con el de la pelota cuando σ se hace más grande que el umbral establecido en la inicialización de *BotTracker*.

5.3.5. Detección múltiple

Un problema que tiene la solución hasta ahora enumerada es que cuando 2 robots colisionan, o un robot está tocando la pelota, no se detecta 1 blob por cada robot o pelota, sino que se detecta un blob único. Entonces es necesario hacer una segmentación de esta única forma.

La detección de robots o pelota en un blob compuesto es una adaptación del método de Graham [3] y funciona de la siguiente forma:

- Se toman los 3 primeros puntos del polígono encontrado.
- Se establece si el punto 3 del conjunto está a la izquierda de la línea formada por los puntos 1 y 2.
- De ser así, podemos cercar una área cuadrada proyectando un cuarto punto p_4 como: $p_1 + (p_3 - p_2)$.
- Se verifica si el cuadrado propuesto tiene un área que corresponda con la de un robot o la pelota. De ser así, se intenta determinar la identidad del robot o de la pelota con esos cuatro puntos.
- Se continúa eliminando el primer punto, agregando un cuarto y se repite el proceso.

Este algoritmo puede encontrar hasta 4 veces el mismo robot, así que solo se agrega un robot siempre que no se haya detectado antes. Lo mismo sucede con la pelota.

En la Figura 9 se puede ver un caso de ejemplo. La zona gris definida por los puntos $p_0, \cdots, p_7$ es el polígono resultante de aplicar `findContours` y `approxPolyDP`.

El algoritmo evalúa los puntos p_0, p_1, p_2 y notando que p_2 está a la izquierda de la línea $\overline{p_0 p_1}$, se proyecta un punto $p'_3 = p_0 + (p_2 - p_1)$. Este punto forma con los 3 anteriores un área potencial para la ubicación de un robot (se muestra en verde sobre la figura).

Después el algoritmo sigue con el conjunto de puntos p_1, p_2, p_3, pero en este caso, p_3 no está a la izquierda de la línea p_1, p_2. De proyectar un punto el área cercada se encontraría fuera del polígono.

Con el caso de p_2, p_3, p_4 se proyecta el punto p'_5 y se forma otra área potencial (se muestra en rojo en la figura).

Para el conjunto p_3, p_4, p_5, se proyecta un p'_6 (no se muestra en la figura) y se obtiene otra potencial área. Sin embargo, esta área coincide en gran parte con la anterior y el robot identificado va a ser el mismo. Este robot se descarta.

El algoritmo continua hasta evaluar todos los conjuntos de puntos (p_i, p_j, p_k) donde, siendo N la cantidad de puntos del polígono, se tiene $i \in [0, N-1]$, $j = (i+1)(mod\ N)$ y $k = (i+2)(mod\ N)$.

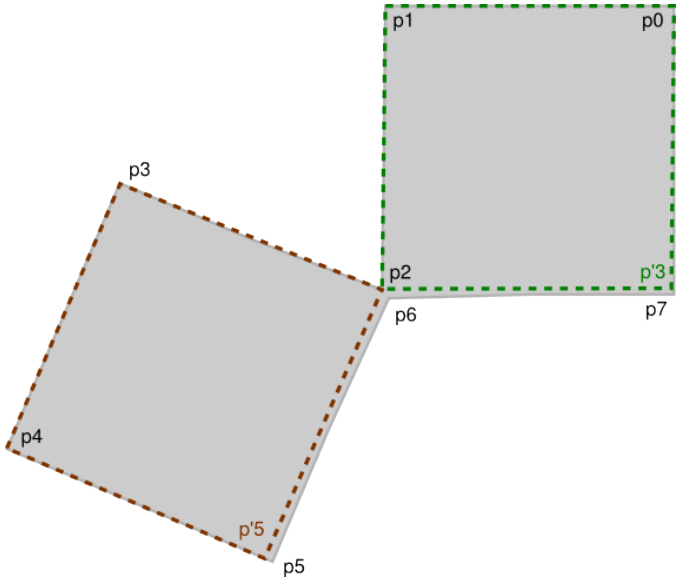

Figura 9: Segmentación de un blob con 2 robots

5.4. Construcción de la muestra

La muestra se representa en 1 objeto de la clase `field_sample` que tiene los siguientes miembros:

- Un objeto de la clase `ball_sample` que representa la pelota.
- Un diccionario (`std::map`) de robots, donde se almacenan objetos de la clase `bot_sample` que representan los robots. La clave del diccionario es el identificador de robot (`e_robot`)
- El timestamp que indica el tiempo en que la muestra era válida.

La muestra se va construyendo en el procesamiento de cada frame. Siempre que se identifica la pelota o un robot, se agregan a la misma. El timestamp se establece al inicio del procesamiento.

Los objetos de tipo `ball_sample` almacenan el centro del área donde se encontró la pelota, la velocidad respecto a la muestra anterior, la forma (polígono donde se encontró la pelota) y el área.

Los objetos de tipo `bot_sample` almacenan el centro del área donde se encontró el robot, la velocidad respecto de la muestra anterior, la forma (polígono donde se encontró el robot), la orientación, el área del robot y el identificador del mismo (`e_robot`).

La posición de robots y pelota es un punto (`cv::Point`). La velocidad es un vector (`cv::Point`) definido como la diferencia de la posición anterior del objeto y la nueva. La orientación del robot es también un vector. La forma es el polígono formado por un conjunto de puntos.

6. Análisis de tiempo de ejecución

Después de múltiples ejecuciones del sistema contra varios casos de pruebas se verifica que el tiempo de ejecución es estable, en el sentido que no hay diferencias de tiempos notables entre varias ejecuciones.

Para un caso en particular, procesando 388 imágenes a 29,97 frames por segundo, se obtuvo un tiempo promedio de procesamiento de cada frame de $4120,92\ \mu s$, con desvío estandar de 845,963 μs. El mínimo fue de 2851,2 μs, mientras que el máximo fue de 6067,74 μs.

Un análisis del tiempo de ejecución de las tareas del procesamiento se muestra en el cuadro 2. Se detalla la cantidad de ejecuciones de cada operación principal junto con el tiempo promedio y total en nanosegundos, y porcentaje del total. Todos estos resultados se obtuvieron sobre una computadora portátil *Dell Inspiron 1545* con *3 GBytes* de memoria RAM y procesador *Intel Core 2 Duo T6500* de 2,10 GHz.

Operación	**Ejecuciones**	**Promedio**	**Total**	**%**
findContours	388	1.704.990	661.534.695	43,59
Conversión a gris	388	1.633.640	633.853.305	41,76
Diferencia absoluta	388	255.943	99.305.868	6,54
approxPolyDP	4088	13.302	54.376.479	3,58
threshold	388	103.878	40.304.634	2,66
try_robot	4835	5.472	26.459.024	1,74
try_ball	466	2.137	995.813	0,07
try_many	146	6.151	898.094	0,06

Cuadro 2: Tiempos de ejecución de operaciones principales.

Se puede ver que algunas tareas se ejecutan con mucha más frecuencia que otras (`try_robot` vs. `findContours`). Sin embargo, se calculó el tiempo total empleado por cada operación sin considerar las repeticiones y se llegó a la conclusión que `findContours` y la conversión del frame a escala de grises son las operaciones con más impacto sobre el tiempo de ejecución (43,59 % y 41,76 % respectivamente), significando más del 85 % del tiempo total de procesamiento.

La operación de conversión de color a escala de grises es la única que opera sobre la imagen completa usando los 3 colores, ya que el resto de las operaciones trabajan ya sea con una sección pequeña de la imagen color o con polígonos.

6.1. Tiempo real

El sistema está escrito en *c++*, usando la librería *OpenCV* que está optimizada para arquitecturas Intel, y una de los objetivos principales de la

misma es la eficiencia.

Además se puso énfasis en minimizar todo tipo de operaciones que puedan causar demoras, como por ejemplo, casi no se usa alocación dinámica de objetos que podría demorar en operaciones de paginación.

Sin embargo, el sistema no corre sobre un sistema operativo de tiempo real duro, y como consecuencia, no existe garantía de que las prestaciones del mismo sean siempre las esperadas.

Bajo condiciones normales, no se detectan latencias excesivas, donde por condiciones normales se entiende que la computadora usada no está saturada con otras tareas, la cantidad de memoria libre es suficiente, y el procesador está con poca carga.

Si se notan demoras cuando el sistema se ejecuta por primera vez, por ejemplo al iniciar la computadora. Se puede ver un lapso pequeño de tiempo en que el sistema no alcanza a procesar los primeros frames con la velocidad necesaria. Pero esta característica no es impedimento, porque poco después de iniciar, el proceso se estabiliza. A parte de esos casos, rara vez se llega a tiempos que excedan los 8 milisegundos. En general cada frame se procesa en 4 milisegundos.

En [6], el sistema de visión demora 4 ms usando imágenes en resolución de $320x240$ y 19 ms en $640x480$. No es posible comparar los tiempos de ejecución contra los de este trabajo, ya que las prestaciones del sistema usado (Pentium 4 a 3,2 GHz) son distintas a las del equipo usado en este trabajo. Pero se puede concluir que los 4 ms logrados deberían ser suficientes. De hecho, pareciera que el tiempo de detección es mínimo respecto de la latencia de la comunicación con los robots por *bluetooth* (39 ms con desvío estandar de 16 ms) que se detalla en [9]. Incluso en [8] se detalla que el tiempo total de ejecución del sistema es del orden de 120 ms, incluyendo captura, detección, estrategia y control.

7. Trabajos relacionados

7.1. División del problema en subproblemas

En este trabajo se propone que la solución al fútbol de robots se puede dividir en tres estadíos bien diferenciados: la detección, la planificación de la estrategía y el comando de los robots. De esas etapas, el trabajo se centra en la primera. Otros trabajos mencionan también divisiones como esta, como en [4]. Incluso en [6] se describe que la detección en si misma es un proceso de siete etapas distintas.

De todos modos, a veces está división puede ser contraproducente, el sistema de reconocimiento puede usar información de los comandos que se le dieron a un robot en la última etapa para decidir que es imposible que ahora se encuentre donde se lo detectó, por mencionar un ejemplo.

7.2. Espacio de color RGB contra espacio HSV ó HSL

En muchos trabajos como [4] o [5] se explica la detección de robots usando el espacio de colores **HSV** y **HSL** en contraste del **RGB**.

En las primeras pruebas realizadas se llegó a la conclusión de que el espacio usado **RGB** era suficiente para una detección con pocas fallas y se continuó con esa estrategia.

Si bien no se hicieron pruebas de las mejoras que podría introducir el uso de los espacios de color mencionados, la teoría de los trabajos mencionados es muy razonable, y se espera que el uso de estos nuevos espacios podría traer mayor estabilidad al reconocimiento.

De todos modos, por la forma en que está escrito el sistema, es fácil agregar conversiones de espacio de color, pero no sin antes determinar el costo en tiempo de ejecución que implica.

7.3. Ajuste de color manual contra ajuste automático

La solución planteada supone que la calibración que se hace al iniciar el procesamiento es estable hasta el final de la competencia. Como consecuencia, se considera que dicha calibración es parte de la configuración inicial. En [9] se sugiere que la configuración inicial de la cámara puede ser un proceso de cerca de una hora.

Algunos trabajos, como [4], [7] o [8] tienen sistemas adaptativos que soportan cambios de iluminación y condiciones de color progresivos, adaptándose a nuevas condiciones.

Incluso en un mismo frame, la captura podría tener colores muy distintos dependiendo de la ubicación del robot por iluminación no homogénea.

Este tipo de problemas no se presentó en las pruebas realizadas, pero no se descarta. Sin embargo, las modificaciones para lograr el reconocimiento adaptativo son sustanciales.

7.4. Aberración de la imagen

No se considera en este trabajo el problema de las aberraciones que produce la cámara. Es de notar, por la forma en que se filmaron los casos de prueba, que los robots tiene áreas muy distintas dependiendo de la ubicación en el campo: los robots más alejados de la cámara tienen áreas más chicas (650 px^2) comparados con los que están más cerca del centro (1200 px^2).

En [4] y [9] se menciona el uso de corrección de los objetos encontrados para eliminar esta anomalía. En general el proceso se hace sobre las formas vectoriales y no directamente sobre la imagen original, por eficiencia. No es difícil de implementar, pero requiere una inicialización del sistema más compleja.

7.5. Detección basada en color contra detección basada en forma

El sistema de detección usado busca formas antes de intentar para determinar posibles robots o pelota. En [4] se usan metodologías que no considera la forma de los blobs, sino que hacen un barrido en busca de colores, y después usan distintas estrategias para reconstruir la forma. Sin embargo, a veces tienen problemas de estabilidad que compensan con algoritmos de seguimiento basados en la historia de reconocimientos anteriores.

Otros enfoques, como [6] usan complejas transformadas para la detección de bordes que pueden llegar a ser costosas. En [9] se sugiere evitar este tipo de detección usando filtros convolucionales.

7.6. Seguimiento de objetos basado en historia

En este trabajo cada uno de los robots de ambos equipos usan un sistema de color en sus "camisetas" para determinar su identidad.

Algunos otros autores tiene estrategias similares, como [9] que usa patrones similares, o como en [6] donde se usan dos círculos de color para identificar los robots.

Es importante notar que en una competencia real no se pueden hacer asunciones respecto de los patrones de color del equipo contrario.

En [4] no se usan patrones: todos los robots del equipo tienen camisetas idénticas. Por lo tanto, si bien hay menos pasos en la detección, es necesario aplicar un método de seguimiento. Las consecuencias de confundir un robot con otro pueden llegar a ser muy problemáticas para el sistema de estrategia, y a veces ese problema recién se corrige cuando hay una pausa en el juego.

8. Posibles mejoras

8.1. Mejoras de estabilidad

Uno de los aspectos importantes del sistema de detección es la estabilidad: que los robots y pelota sean identificados siempre, sin confundirlos, y con la mayor precisión posible de los detalles de posición, velocidad y orientación. El sistema implementado puede en ocasiones tener problemas, donde por lo general están causados por pobre calidad de iluminación o captura de la imagen, interferencia de elementos inesperados (por ejemplo cuando aparece la mano del operador para ajustar la posición de los robots o pelota), colisión de robots entre si o con la pelota.

Respecto de la imagen, se puede mejorar usando cámaras más precisas, con calibración de color mejor. Incluso el uso de buenas fuentes de iluminación, ya que en las muestras creadas para el proyecto se usaron lamparas que no eran las más adecuadas.

También se generan problemas cuando aparecen figuras externas. Sin embargo, las mismas solo suceden cuando el juego está pausado, y el sistema se recupera rápido una vez que la interferencia desaparece.

El punto más crítico es cuando los robots colisionan. En la implementación actual, se usa un algoritmo derivado del método Graham para calcular el *Convex Hull* [3], y el algoritmo se aplica solo sobre los blobs que no resultaron en robots o pelotas en un principio. Este método todavía tiene fallas.

Para resolver estos casos poco típicos se pueden usar varias estrategias:

- Agregar otros algoritmos a los blobs, por ejemplo después de pasar el derivado de Graham. Incluso se puede modificar la librería para que tenga una batería de estrategias de detección.
- Buscar usando barrido de color. Es una estrategia que usan otros autores.
- Usar información histórica. El sistema va acumulando información de detección en su historial (*fields*). Se pueden usar estos datos para establecer posiciones nuevas, aunque siempre hay que comprobar si la estimación de nuevas posiciones tienen los objetos buscados.

Cualquiera de estas alternativas se pueden implementar, de hecho todas, sin modificar mucho la interfaz. La mayor consideración al respecto recae sobre el impacto en tiempo de ejecución. Sin embargo, estas estrategias se aplican sobre un espacio reducido de la imagen, incluso sobre las formas geométricas ya detectadas, donde las operaciones más complejas no deberían de tener un impacto mayor considerando los costos de las operaciones sobre la imagen completa.

8.2. Mejoras de rendimiento

Como se ha detallado anteriormente en 6, hay dos operaciones principales que producen la mayor demora: la conversión a escala de grises, y la detección de bordes usando `findContours`.

Las operaciones de matrices como la conversión a escala de grises, la diferencia absoluta y la umbralización son operaciones que trabajan procesando pixeles de la imagen en forma independiente y son casos interesantes para optimizar usando paralelismo. Son del tipo de operaciones *Embarrassingly parallel* [13], donde practicamente el tiempo de procesamiento podría acelerarse en el orden de la cantidad de procesadores usados. Y se puede decir que existe una tendencia a la computación paralela, como se enumera en [14]:

> Como el consumo de alimentación (y consecuentemente la generación de calor) por computadoras se ha convertido en una preo-

cupación en los años recientes, el cómputo paralelo se ha convertido en el paradigma predominante en arquitecturas de computador, mayoritariamente en la forma de procesadores *multi-core*.

Incluso el sistema usado para el desarrollo de este trabajo es una computadora portátil de doble nucleo, donde aplicar computo paralelo solo en la conversión de imágenes de color a blanco y negro podría reducir el tiempo a cerca de la mitad[3] , llevando el tiempo promedio de los aproximadamente $1633\mu s$ actuales a $820\mu s$, lo que reduciría el tiempo total de procesamiento por frame de los $4120\mu s$ a cerca de $3300\mu s$, esto es, casi un 20 % menos.

8.3. Generalización a otras ligas

Este trabajo estuvo limitado a una sola categoría de las competencias de fútbol de robots. Sin embargo, se puede modificar fácilmente para soportar ligas donde se juegan con 11 robots, o con 3 (liga grande y liga chica).

Incluso sería posible modificarla para que el tipo de liga sea otro de los parámetros de configuración del sistema.

9. Conclusiones

El trabajo presenta un sistema de visión por computador para un juego de fútbol de robots, donde la preocupación motivante es el tiempo de procesamiento.

Se logró un sistema confiable, con tiempos de ejecución comparables a los encontrados en literatura de referencia.

La librería construida usa pocos recursos y debería funcionar sin inconvenientes en equipos de bajo costo. Además es fácilmente agregable[4] a un sistema completo que incluya el resto de las operaciones necesarias para un sistema de fútbol de robots.

Si bien todo el proyecto se construyó usando maquetas de cartón, se hizo una prueba usando un video obtenido del sitio de la FIRA [15] del partido final de la competencia llevado a cabo en el año 2010 donde se puede ver que el sistema debería funcionar en la competencia real. En la Figura 10 se muestra la detección de formas mencionada, correspondiente a la liga chica (3 robots por equipo). El sistema detecta sin problemas el área de los robots y la pelota (se muestra en blanco) aún cuando el video original es de baja calidad.

[3] No se está considerando el tiempo de sincronización de las tareas, aunque el mismo debería ser despreciable.

[4] agregable del inglés *embeddable*

Figura 10: Pruebas del sistema en una competencia real

Referencias

[1] Topological Structural Analysis of Digitized Binary Images by Border Following.

Satoshi Suzuki and Keiichi Abe.

[2] Ramer–Douglas–Peucker algorithm.

http://en.wikipedia.org/wiki/Ramer-Douglas-Peucker_algorithm

[3] Método de Graham.

http://en.wikipedia.org/wiki/Graham_scan

[4] An Examplary Robot Soccer Vision System.

Norman Weiss, Lars Hildebrand. Department of Computer Science, University of Dortmund, Alemania.

CLAWAR/EURON Workshop on Robots in Entertainment, Leisure and Hobby December 2 – 4, 2004, Vienna, Austria

[5] An Omnidirectional Vision System for Soccer Robots António J. R. Neves, Gustavo A. Corrente y Armando J. Pinho

Departamento de Electrónica e Telecomunicações - IEETA Universidade de Aveiro, 3810–193 Aveiro, Portugal

[6] Towards Model-based Vision Systems for Robot Soccer Teams.

Murilo Fernandes Martins, Flavio Tonidandel and Reinaldo A. C. Bianchi Centro Universitário da FEI, Brazil.

Fuente: Robotic Soccer, libro editado por: Pedro Lima, ISBN 978-3-902613-21-9, pp. 598, Diciembre 2007, Itech Education and Publishing, Vienna, Austria

[7] A Real-Time Auto-Adjusting Vision System for Robotic Soccer.

Matthias Jüngel, Jan Hoffmann, Martin Lötzsch Institut für Informatik, LFG Künstliche Intelligenz, Humboldt-Universität zu Berlin, Rudower Chaussee 25, 12489 Berlin, Alemania

[8] Robust Adaptive Vision for Robot Soccer.

Gordon Wyeth and Ben Brown. Computer Science and Electrical Engineering, University of Queensland, Brisbane, Australia 4072

[9] The Secrets of Robot Football (Seminario).

Joerg C. Wolf Centre for Robotics and Intelligent Systems. University of Plymouth, Drake Circus, Plymouth, U.K.

[10] Computer Vision. A modern approach.

David A. Forsyth, Jean Ponce. Agosto del 2002. ISBN-10 0130851981 ISBN-13 978-0130851987

[11] Computational Geometry. Algorithms and Applications. Third Edition.

Mark de Berg, Otfried Cheong, Marc van Kreveld, Mark Overmars. ISBN 978-3-540-77973-5

[12] Learning OpenCV.

Gary Bradski and Adrian Kaehler. Primera edición, Septiembre de 2008. Publicado por O'Reilly Media, Inc. ISBN: 978-0-596-51613-0

[13] Embarrassingly parallel en Wikipedia.

http://en.wikipedia.org/wiki/Embarrassingly_parallel

[14] Parallel computing en Wikipedia.

http://en.wikipedia.org/wiki/Parallel_computing

[15] FIRA - Federation of International Robot-soccer Association

http://www.fira.net/

[16] Sitio de la RoboCup

http://www.robocup.org/