Un Sistema de Visión Global para Fútbol de Robots Físicos

Guillermo Torres y Eduardo Grosclaude

Departamento de Ingeniería de Computadoras, Facultad de Informática.
Universidad Nacional del Comahue
Buenos Aires 1400, Neuquén Capital, Argentina. CP 8300
{guillermo.torres, eduardo.grosclaude}@fi.uncoma.edu.ar

Resumen. El desarrollo de un partido de Fútbol de Robots físicos tiene tres aspectos fundamentales: los robots, que consisten en la electrónica, partes mecánicas y parches; el software de control desarrollado por cada equipo, que determina el comportamiento de los robots; y el sistema de visión, que debe identificar los objetos y entregar su posición y orientación al software de control. En el presente trabajo desarrollamos y evaluamos un framework basado en plugins que nos permita construir, como aplicación, un Sistema de Visión Global (SVG) para Fútbol de Robots físicos. El *framework* ha sido desarrollado para ser utilizado en un ámbito educativo, donde pueda ser utilizado como marco introductorio a la problemática de Visión por Computadoras.

Palabras claves: framework - servidor de video - fútbol de robots

1. Introducción

En el fútbol de robots (FR) se integran las Ciencias de la Computación y la tecnología robótica, y se motiva la investigación y desarrollo de soluciones a problemas clásicos y nuevos en un dominio por todos conocido. Los equipos de FR pueden medirse jugando partidos en eventos anuales, organizados a nivel nacional por el CAFR (Campeonato Argentino de Fútbol de Robots), y a nivel internacional por FIRA (Federation of International Robot-soccer Association)[1] o RoboCup (Robot World Cup)[2]. Las competencias de FR se diferencian principalmente por ser **simuladas** o **físicas**, y de **visión local** o **global**. Además, las competencias poseen categorías con reglas específicas que establecen la duración de un partido, dimensiones del campo de juego, cantidad y dimensiones de los robots, etc.

[1] `http://www.fira.net`
[2] `http://www.robocup.org`

En la competencia **simulada** es un software el encargado de representar virtualmente a cada robot, pelota, y el campo de juego. Mientras que, en la competencia denominada **física,** los objetos son físicamente existentes.

En las competencias de **visión local**, se ubica una cámara en cada robot. Lo que se obtiene es una visión parcial del campo de juego desde cada robot. Las cámaras tienen una ubicación dinámica, y no todos los objetos son capturados por cada cámara. Para el caso de **visión global**, se ubica una cámara sobre el punto medio de la cancha de manera estática, capturando todo el campo de juego.

El desarrollo de un partido de FR tiene tres aspectos fundamentales: **los robots**, que consisten en la electrónica, partes mecánicas y parches; el **software de control desarrollado por cada equipo**, que determina el comportamiento de los robots; y **el sistema de visión**, que debe identificar los objetos (parches y pelota) y entregar sus datos (posición y orientación) a dicho software de control. Con los datos que recibe desde el sistema de visión, el software de control determina las acciones que los robots deben ejecutar, y se las comunica por vía inalámbrica.

Diversos sistemas de visión global han sido desarrollados para el FR físicos [1, 3, 11, 9]. Los trabajos presentados en [1, 3, 11] resultan difíciles de configurar y requieren una cámara de video de alta calidad. En todos los casos, la captura y procesamiento de imágenes están acopladas en una misma unidad de ejecución, y aunque son de código abierto, su modificación o reusabilidad implica un estudio casi completo.

En el presente trabajo desarrollamos un **framework basado en plugins** que nos permita construir, como aplicación, un **Sistema de Visión Global (SVG)** para Fútbol de Robots Físicos en la categoría Senior del CAFR. Este sistema debe ser funcional y fácil de configurar, utilizable con bajos requerimientos de hardware, capturar y procesar imágenes en forma desacoplada, ser de código abierto, y facilitar la reusabilidad de los algoritmos de Visión por Computadoras (VC). El *framework* ha sido desarrollado para ser utilizado en un ámbito educativo, donde pueda ser utilizado como marco introductorio al tema mediante la experimentación de los distintos algoritmos propios de VC.

El resto de este trabajo está estructurado de la siguiente manera. En la sección 2 se describe la arquitectura del *framework* y se aborda el caso de uso de un sistema de visión global para fútbol de robots físicos de la categoría Senior del CAFR. Aquí instanciaremos el *framework* con la implementación de los diferentes *plugins* que permiten identificar la posición y orientación de los parches, y la posición de la pelota. En la sección 3 se presentan los resultados experimentales y su análisis. Finalmente, la Sección 4 resume las principales conclusiones que se han obtenido y las perspectivas a futuro.

2. Un framework para construir un SVG

El esquema de la Figura 1 muestra la arquitectura del *framework* basado en *plugins* que hemos desarrollado. Se distinguen cuatro grandes componentes: el

hilo principal, que es responsable de la Interfaz Gráfica de Usuario (GUI), incluyendo todos los diálogos de visualización y configuración; el **hilo de captura**, constituido por el módulo de captura de cuadros, que es responsable de obtener las imágenes desde un origen, tal como un dispositivo de video (por ejemplo, una *webcam*) o un archivo de video; los **hilos de procesamiento**, cada uno responsable de ejecutar un conjunto ordenado de plugins, organizados en una *pila de plugins;* y el **buffer circular**, que gestiona el almacenamiento de los cuadros capturados para su posterior procesamiento.

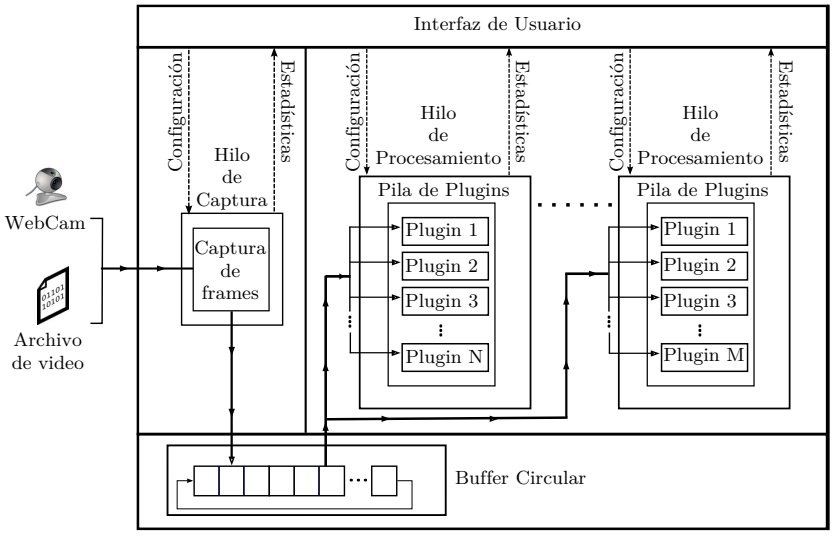

Figura 1: Esquema funcional del SVG

El *framework* ha sido desarrollado en modo *multi-threaded*, con lo cual los hilos de procesamiento son ejecutados en paralelo en aquellas plataformas cuya arquitectura así lo permita. El tratamiento de los datos responde al **modelo productor/consumidor**, donde el hilo de captura produce cuadros y los hilos de procesamiento los consumen (respectivamente, agregando y quitando cuadros del *buffer* circular). Cuando se consume un cuadro desde el *buffer* circular, éste debe ser procesado por todos los hilos de procesamiento. Los hilos se coordinan encontrándose frente a una construcción *barrier* al final del ciclo de consumo de un cuadro. El buffer circular implementa además la exclusión mutua de los hilos sobre cada cuadro. El buffer circular puede configurarse para funcionar en dos modos: forzando el procesamiento de todos los cuadros capturados, o prefiriendo la obtención del cuadro más recientemente capturado a costa de descartar cuadros anteriores.

Los puntos de extensión para el cliente del framework son los plugins. El desarrollador define la granularidad de cada *plugin* para implementar los algoritmos de VC. La pila facilita el intercambio de plugins para evaluarlos. Así, la experimentación no requiere la construcción de una aplicación completa por

cada configuración de plugins que se plantee.

El framework ha sido desarrollado con C++, Qt, OpenCV[3]+Intel TBB (Intel® Threading Building Blocks).Un caso de uso: Fútbol de Robots Físicos

La aplicación del *framework* que estudiamos es un **Sistema de Visión Global (SVG)** para Fútbol de Robots Físicos en la categoría Senior del CAFR. En esta categoría las dimensiones de la cancha son de 2200 x 1500 mm. Cada equipo es integrado por tres robots, de dimensiones no superiores a 30 cm de diámetro, ocupando uno de ellos el puesto de arquero. Cada robot incluye un receptor inalámbrico que le permite recibir los comandos a ejecutar. En la parte superior de cada robot se ubica un parche o sombrero con colores para identificarlo. El formato del parche es representado en la figura 2. Allí, el color del cuadrado central, que denominamos región principal, indica el equipo, y los colores de los cuatro círculos menores, que denominamos regiones secundarias, codifican de manera unívoca a cada parche, identificando a cada unidad en juego.

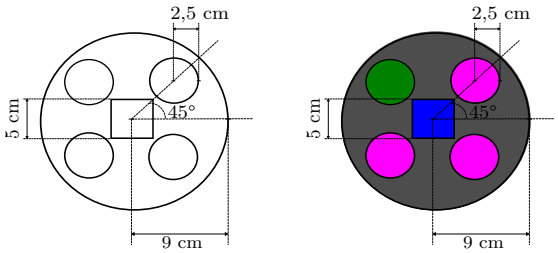

Figura 2: Formato (izquierda) y ejemplo (derecha) de un parche.

Captura de cuadros

Para la experimentación se utiliza una *webcam* como dispositivo de captura de bajo costo. La captura de cuadros es administrada por el *framework*. La cámara entrega 30 fps, con lo cual el SVG dispone de 33 ms para el procesamiento de cada cuadro en forma concurrente por todos los hilos.

Procesamiento de cuadros

Para este caso de uso hemos decidido procesar los cuadros utilizando dos hilos de procesamiento de cuadros que llamaremos A y B. El hilo de procesamiento A se dedica a identificar a los parches, buscando primero el color de la región principal, y luego buscando a su alrededor los colores de las regiones secundarias. El hilo de procesamiento B queda dedicado a identificar a la pelota por su color. Cada hilo de procesamiento cuenta con su propia pila de *plugins* (descriptos en detalle a continuación) como muestra el Cuadro 1.

[3] `http://opencv.org`

Hilo A (parches)
CALIBRACIÓN
CONVERSIÓN DE COLOR
SEGMENTACIÓN DE COLOR
MORFOLOGÍA
REGIÓN PRINCIPAL
DETECCIÓN DE PARCHES
DIFUSIÓN

Hilo B (pelota)
CALIBRACIÓN
CONVERSIÓN DE COLOR
SEGMENTACIÓN DE COLOR
MORFOLOGÍA
DETECCIÓN DE PELOTA
DIFUSIÓN

Cuadro 1: Pila de *plugins* de cada hilo de procesamiento

Descripción de los Plugins

Todos los plugins [8] han sido desarrollados usando C++ y OpenCV. En particular, las siguientes características son compartidas por los *plugins* de REGIÓN PRINCIPAL, DETECCIÓN DE PARCHES y DETECCIÓN DE PELOTA:

- Utilizan la biblioteca cvBlob[4] para obtener el área y centro de las regiones,
- Aplican una matriz homográfica para obtener la posición en el mundo real (en milímetros) a partir de la posición de un objeto en la imagen (en píxeles). Para calcular esta matriz, se requiere, por única vez y previamente al uso del *plugin*, que se indique la ubicación de los píxeles en la imagen de las cuatro esquinas, y las dimensiones en milímetros, del campo de juego.

Plugin de CALIBRACIÓN

El objetivo de este *plugin* es realizar correcciones a deformaciones en la imagen introducidas por la propia cámara. Utiliza como entrada una imagen en el espacio de color RGB, y los parámetros y coeficientes de calibración de la cámara. Estos dos últimos deben obtenerse previamente al uso de este *plugin*.

Para obtener esos parámetros y coeficientes de calibración se usa la aplicación de calibración provista por OpenCV[5]. Esa aplicación utiliza un método de calibración basado en [10], y durante el proceso usa un patrón con el aspecto de un tablero de ajedrez[6]. Con este paso previo obtenemos los parámetros intrínsecos y extrínsecos, y los coeficientes de distorsión radial y tangencial de la cámara. Con esos datos, el *plugin* aplica una transformación geométrica al cuadro capturado para alcanzar su objetivo.

Plugin de CONVERSIÓN DE COLOR

El objetivo de este *plugin* es convertir una imagen de entrada a su imagen equivalente en el espacio de color HSV. Utiliza como entrada una imagen en

[4] `http://code.google.com/p/cvblob/`

[5] `http://docs.opencv.org/_downloads/camera_calibration.cpp`

[6] Por las características del objeto de calibración, este método es clasificado como calibración basada en el plano 2D [4].

el espacio de color RGB. A diferencia del espacio de color RGB, el espacio de color HSV desacopla la intensidad del color. Esto nos permite trabajar sobre el matiz (H), sin que la variabilidad de iluminación afecte significativamente el color de cada píxel en la imagen [7]. Este es el principal motivo por el que hemos seleccionado el espacio de color HSV. El *plugin* implementa la conversión del espacio de color RGB al HSV sobre cada píxel de la imagen de entrada.

Plugin de SEGMENTACIÓN

El objetivo de este *plugin* es extraer del cuadro sólo los colores usados por los parches. Utiliza como entrada una imagen, y para segmentar los colores se definen umbrales multidimensionales, ambos en el espacio de color HSV. Antes de usar este *plugin*, es necesario ajustar los rangos de valores para cada componente H, S, y V de cada color. En este *plugin* hemos implementado el método de segmentación por umbralización veloz de [2]. Al instanciarse el *plugin*, con la definición de los umbrales multidimensionales hechos previamente, se construyen los arreglos combinados de colores para los colores usados en los parches.

Adicionalmente, por cada color usado en algún parche, se genera un cuadro de la misma resolución que el cuadro de entrada pero en blanco y negro. Cada uno de esos cuadros generados representa un color a ser extraído desde el cuadro de entrada y es inicializado en negro antes de comenzar a procesar el cuadro de entrada.

El proceso consiste en recorrer cada píxel (x, y) del cuadro de entrada y determinar mediante la operación binaria AND usando los arreglos combinados de colores a qué clase de color pertenece. De acuerdo a la clase de color a la que pertenece el píxel (x, y), se escribe con blanco esa coordenada en su correspondiente imagen binaria. Lo que resulta en que cada imagen binaria posea las regiones segmentadas del color que representa.

Plugin de MORFOLOGÍA

El objetivo de este *plugin* es disminuir la cantidad de huecos internos en las regiones de una imagen. Utiliza como entrada las imágenes binarias correspondientes a las regiones principales de ambos equipos. El *plugin* aplica la operación morfológica de apertura [6] sobre las imágenes de entrada para alcanzar su objetivo.

Plugin de REGIONES PRINCIPALES

El objetivo de este *plugin* es hallar y obtener el área de las regiones principales de ambos equipos. Utiliza como entrada las imágenes binarias que representan las regiones principales.

El proceso consiste primero en etiquetar las regiones, obtener su área, y luego filtrar por su tamaño de área para descartar las regiones demasiado pequeñas o grandes. Luego, con los datos extraídos de las regiones resultantes, se conforman las propiedades de las regiones principales, que utilizaremos en el plugin de detección de parches.

Plugin de DETECCIÓN DE PARCHES

El objetivo de este *plugin* es determinar si las regiones secundarias que se hallan alrededor de una región principal coinciden con la codificación de un parche. Utiliza como entrada las propiedades de las regiones principales, las imágenes binarias de las regiones secundarias, y un área cuadrada en píxeles. Con el área cuadrada, el *plugin* determina un área reducida de procesamiento alrededor del centro de cada región principal. Luego procesa cada imagen binaria de una región secundaria hallada dentro del área reducida, de acuerdo a los colores usados por cada equipo. En caso de hallar regiones secundarias, para cada una se extrae su área, centro de área, y ángulo. Para obtener el ángulo, primero, trasladamos el origen del eje de coordenadas (x, y) al centro de la región principal. Luego, para ese eje de coordenadas, calculamos el ángulo formado por la recta que atraviesa el centro de la región principal y el centro de la región secundaria.

A continuación, se ordenan por ángulo de menor a mayor las regiones. Con este ordenamiento, se representa una cadena para aplicar un método estructural de correspondencia por cadena [5]. La región principal siempre tiene un ángulo de 0 grados; por ello, siempre es tomada como punto inicial de la cadena. De esta manera, cada cadena obtenida desde su área reducida, es desplazada circularmente y evaluada con cada una de las cadenas que describen a un parche para determinar si se igualan. En caso de igualarse, se marca al parche en la imagen y se actualizan sus datos. En caso contrario, la combinación hallada se descarta.

Plugin de DETECCIÓN DE PELOTA

El objetivo de este *plugin* es detectar en la imagen la pelota. Utiliza como entrada la imagen binaria que representa el color de la pelota. De la imagen de entrada se extraen el área y centro de cada región. Luego se filtra por tamaño de área para descartar las regiones demasiado pequeñas o grandes, y se selecciona la región de mayor área como pelota. En caso de hallar la pelota, se actualiza su posición en píxeles y en milímetros.

Plugin de DIFUSIÓN

Difunde a ambos equipos los datos de los objetos a través de la red LAN, por broadcast. Los datos de los parches incluyen la posición (x, y) en píxeles y en milímetros, y la orientación en grados. Los datos de la pelota incluyen la posición (x, y) en píxeles y en milímetros. Para todos los objetos, se incluye una bandera, que indica si ha sido identificado el objeto en el cuadro en curso, y número de cuadro. En caso de no ser identificado un objeto en el cuadro en curso, se difunden los datos del objeto de la última vez que fue identificado.

3. Experimentación

La experimentación está orientada a evaluar la velocidad y precisión del SVG. El SVG está configurado con los *plugins* descritos anteriormente. La plataforma usada fue un equipo con microprocesador Intel® Core 2 Duo de 1.40 GHz, 3 GB de RAM, instalado con GNU/Linux Ubuntu 10.04, y una *webcam* USB QuickCam© Deluxe for Notebooks de Logitech. Se usó un campo de juego de 2200x1500 mm, 6 parches, y una pelota. La *webcam* se ubicó sobre el punto central del campo de juego a 2,30 metros de altura, y se configuró en el SVG para capturar cuadros de una resolución de 352x288 píxeles.

Para la experimentación se tomó una muestra de 301 cuadros consecutivos, entre el cuadro número 600 y 900, y se recogieron los datos de los objetos asociados a esa muestra y los tiempos totales de ejecución de los hilos.

En la Fig. 3 se aprecian los tiempos de ejecución obtenidos para la muestra. El tiempo de ejecución de ambos hilos de procesamiento es inferior al del hilo de captura el 96,35 % de las veces. Esto quiere decir que cada cuadro es procesado por los hilos de procesamiento antes de que esté disponible el próximo cuadro.

En el caso contrario, si el ciclo de alguno de los hilos de procesamiento superara el tiempo de captura de cuadros, y si esta situación se mantuviera en cuadros sucesivos, se estarían capturando más cuadros de los que se pueden procesar. Para este caso, el SVG está configurado para descartar cuadros y que siempre se obtenga el cuadro más recientemente capturado desde el buffer circular.

La Fig. 3 muestra que el 3,65 % de las veces el tiempo de procesamiento del hilo A es superior al del hilo de captura. Sin embargo, la diferencia entre tiempos no se conserva durante un número de capturas suficiente como para generar una situación de cuadros descartados.

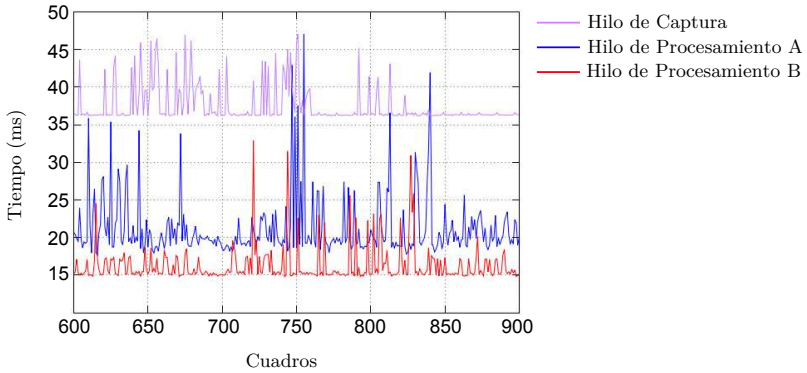

Figura 3: Tiempo de consumo por cuadro para el hilo de captura, e hilos de procesamiento A y B.

Para la misma muestra, los registros de los parches se comportan similarmente, por lo cual seleccionamos sólo el parche #1 para mostrar los resultados de la experimentación. Es importante señalar que durante la experimentación realizada, la identificación, posición *(x,y)* en mm, y orientación, entregados por el

SVG coincidieron con la identificación, posición en mm y orientación verdadera en el campo de juego.

El parche #1 fue identificado 299 veces y sólo 2 veces no se lo identificó. Por su parte, la pelota, que se mantuvo siempre completamente visible, fue detectada 300 veces y no fue detectada 1 vez. Lo que resulta en que el software de cada equipo obtenga la ubicación de los objetos en el campo de juego la mayor parte del tiempo.

La Fig. 4 a) muestra cómo varía la posición central del parche #1, entre un cuadro y su consecutivo, en la imagen (píxeles) y en el mundo real (milímetros). Aquí se observa que la posición central puede variar en una distancia de hasta $1,42$ píxeles en la imagen, y $8,49$ mm en el mundo real. Para el software de los equipos que reciben estos datos, significa que el SVG entrega la posición (x, y) de los parches con un error inferior a 1 cm.

La Fig. 4 b) muestra cómo varía la diferencia en orientación percibida del parche #1 entre un cuadro y su consecutivo cuando se encuentra inmóvil. La variación se concentra alrededor de los 5°, con 8° en el peor caso (2 % de las veces).

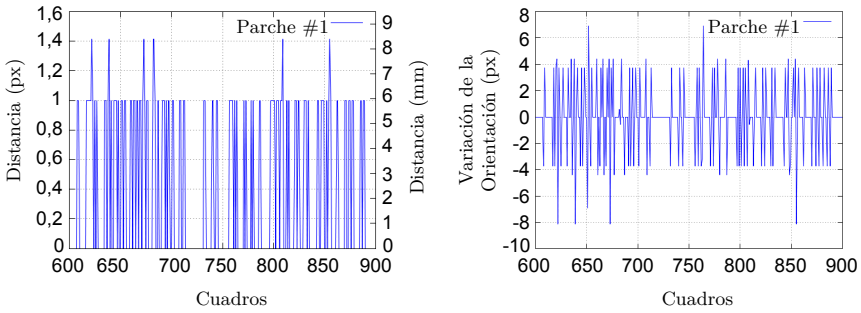

Figura 4: a) Variación de la posición central del parche #1. b) Variación de la orientación del parche #1.

4. Conclusiones y trabajos futuros

Los resultados de la evaluación experimental muestran que el SVG cumple de manera satisfactoria con los requerimientos de tiempo y posee una precisión aceptable para determinar la identificación, posición y orientación de los objetos en el campo de juego. Por ello decimos que es viable su uso en el campeonato de fútbol de robots físicos de la categoría Senior.

También consideramos que el framework desarrollado puede ser usado como herramienta introductoria en un curso de VC o para la construcción de aplicaciones de VC. La posibilidad de capturar de cuadros desde una webcam sencilla de bajo costo contribuye a poner esta herramienta al alcance de los estudiantes e investigadores.

El framework y los plugins vistos en el caso de uso admiten mejoras. Algunas tareas pendientes son: incorporar el uso de XML para la configuración general (por ej., hilos de procesamiento, sus pilas y plugins) para automatizar la experimentación; agregar la opción de múltiples fuentes de captura para visión estereoscópica; dar tratamiento en el plugin correspondiente a la oclusión de la pelota cuando algún robot la oculta; agregar otras técnicas de reconocimiento de objetos.

Referencias

[1] Jacky Baltes. Doraemon: Object orientation and id without additional markers. In *2nd IFAC conference on mechatronic systems*, pages 845–850, 2002.

[2] James Bruce, Tucker Balch, and Manuela Veloso. Fast and inexpensive color image segmentation for interactive robots. In *Intelligent Robots and Systems, 2000.(IROS 2000). Proceedings. 2000 IEEE/RSJ International Conference on*, volume 3, pages 2061–2066. IEEE, 2000.

[3] Paul Furgale, John Anderson, and Jacky Baltes. Real-time vision-based pattern tracking without predefined colors. In *Proceedings of the Third International Conference on Computational Intelligence, Robotics, and Autonomous Systems (CIRAS)*, 2005.

[4] Gerard Medioni and Sing Bing Kang. *Emerging topics in computer vision*. Prentice Hall PTR, 2004.

[5] Gonzalo Pajares Martinsanz and Jesus M. de la Cruz Garcia. *Visión por computador*. Alfaomega RA-MA, 2d. edition, 2008.

[6] L. Shapiro and G. Stockman. *Computer Vision*. Prentice Hall, 2000.

[7] Milan Sonka, Vaclav Hlavac, and Roger Boyle. *Image processing, analysis, and machine vision*. Thompson Learning, 2008.

[8] Guillermo Eduardo Torres, Javier Balladini, and Rodolfo Del Castillo. Un Sistema de Visión Global para Fútbol de Robots. *WICC 2012*, Mar. 2012.

[9] Gonzalo Zabala, Sebastián Blanco, Ricardo Morán, and Matías Teragni. Servidor de video para fútbol de robots físico desarrollado bajo opencv y microsoft visual studio. In *XVI Congreso Argentino de Ciencias de la Computación*, 2010.

[10] Zhengyou Zhang. A flexible new technique for camera calibration. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 22(11):1330–1334, 2000.

[11] Stefan Zickler, Tim Laue, Oliver Birbach, Mahisorn Wongphati, and Manuela Veloso. Ssl-vision: The shared vision system for the robocup small size league. In *RoboCup 2009: Robot Soccer World Cup XIII*, pages 425–436. Springer, 2010.