UNIVERSIDAD DE CHILE
Facultad de Ciencias Físicas y Matemáticas
Departamento de Ingeniería Eléctrica

DETECCIÓN Y SEGUIMIENTO DE ROBOTS ARTICULADOS MEDIANTE ANÁLISIS COMPUTACIONAL DE IMÁGENES

TESIS PARA OPTAR AL GRADO DE MAGÍSTER EN CIENCIAS DE LA INGENIERÍA MENCIÓN ELÉCTRICA

MEMORIA PARA OPTAR AL TITULO DE INGENIERO CIVIL ELECTRICISTA

MATIAS ESTEBAN ARENAS SEPÚLVEDA

PROFESOR GUÍA:
JAVIER RUIZ DEL SOLAR SAN MARTÍN

MIEMBROS DE LA COMISIÓN:
RODRIGO PALMA BEHNKE
RODRIGO VERSCHAE TANNENBAUM
PABLO ZEGERS FERNÁNDEZ

Santiago de Chile
Abril 2009

AGRADECIMIENTOS

El presente trabajo fue financiado parcialmente por el proyecto FONDECYT 1061158.

"Creo que no nos quedamos ciegos, creo que estamos ciegos, Ciegos que ven, Ciegos que, viendo, no ven" – "Ensayo sobre la ceguera" - Saramago.

RESUMEN DE LA MEMORIA
PARA OPTAR AL GRADO DE
MAGÍSTER EN CIENCIAS DE LA INGENIERÍA
MENCIÓN INGENIERÍA ELÉCTRICA
Y AL TÍTULO DE
INGENIERO CIVIL ELECTRICISTA
POR: MATIAS ARENAS SEPULVEDA
FECHA: 02 DE MARZO DE 2009
PROF. GUIA: DR. JAVIER RUIZ DEL SOLAR

"DETECCIÓN Y SEGUIMIENTO DE ROBOTS ARTICULADOS MEDIANTE ANÁLISIS COMPUTACIONAL DE IMÁGENES"

El objetivo de esta tesis es diseñar e implementar un sistema de detección y seguimiento de robots articulados mediante el análisis computacional de imágenes. El principal aporte es extender el uso de técnicas de detección de objetos rígidos a robots articulados y lograr un seguimiento en tiempo real de la posición de éstos en imágenes y secuencias de video. El trabajo realizado está enfocado principalmente a condiciones de fútbol robótico pero es fácilmente extendible a otras aplicaciones.

El reconocimiento de robots articulados requiere la detección de éstos en cualquier estado y posición. Para lograrlo, se probaron distintos tipos de clasificadores de manera de obtener una mayor tasa de detección, independiente del estado del robot (acostado, parado, caminando, etc…). Los robots utilizados para probar los algoritmos fueron el robot Aibo y los robots de tipo Humanoide (basado en el modelo Hajime).

El sistema antes mencionado necesitó la creación e implementación de distintas herramientas computacionales. Para la detección de robots en imágenes se utilizó un sistema basado en Adaboost, y para el posterior seguimiento se ocupó el algoritmo "mean-shift". Para la detección mediante Adaboost se generaron una serie de bases de datos para el entrenamiento del algoritmo. Posteriormente se construyeron diversos clasificadores (frontal, lateral, trasero, global, etc…), y se probaron distintas estrategias de detección.

El detector con mejores resultados para Aibos fue el Lateral, con una tasa de detección de hasta 94.7% con 98 falsos positivos en la base de datos AIBODetUChileEval. Luego siguen el detector Trasero, con 89.9% y 166 falsos positivos y, por último, el detector Frontal con 89.4% y 254 falsos positivos. Finalmente se probó la detección de los Aibos en todas las posiciones con un detector múltiple, el cual obtuvo una tasa de detección de 94.8% con 392 falsos positivos. Aplicando solo el detector frontal sobre todas las imágenes con Aibos obtuvo solo un 90% de detecciones con 392 falsos positivos, pero es más rápido que el detector múltiple. Para los Humanoides se desarrolló un solo detector que logró un 92.2% de detecciones con 123 falsos positivos sobre la base de datos HDetUChileEval.

Se concluyó finalmente que los clasificadores Adaboost elegidos en este trabajo para hacer las clasificaciones reportan excelentes resultados, y no se duda que puedan también hacerlo en otros tipos de aplicaciones de similares características. Además se logró hacer la detección y el seguimiento de robots en tiempos muy cercanos al tiempo real, lo cual permite ocuparlo en aplicaciones con altas restricciones de procesamiento.

INDICE DE CONTENIDO

1. **Introducción**

Para los seres humanos la vista es uno de los sentidos más importantes. La visión permite identificar objetos, examinarlos sin tocarlos, determinar tamaños y relaciones espaciales y moverse con seguridad en distintos entornos. Estas mismas habilidades se pueden implementar en distintos tipos de máquinas y robots, y cobran aun más importancia en la robótica móvil. A medida que avance la tecnología los robots deberán crear o tener un mapa de su entorno, moverse, y poder reconocer e interactuar con objetos que se encuentren a su alrededor.

Las tareas de visión computacional son relativamente nuevas en la historia de la tecnología, y sólo en los últimos 15 años han comenzado a desarrollarse con mayor intensidad. La principal limitación que existía antes era la capacidad de procesamiento de los computadores, pero hoy, con el constante avance de la tecnología, esta barrera ha sido largamente superada. Cada día se crean nuevas herramientas y productos que emulan de mejor manera las habilidades humanas (no solo de visión). En la actualidad, ya se pueden ver en el comercio cámaras que leen patentes de automóviles, miden tamaños de objetos, o detectan caras, entre otras funciones. Sin embargo, aún hay muchas dificultades para la detección de objetos móviles que cambian de forma.

En el contexto de la robótica móvil, es muy útil detectar e identificar objetos dado que para interactuar con éstos, primero hay que reconocerlos. Esto se hace aún más presente en el caso del fútbol robótico donde los robots deben poder interactuar entre ellos y con otros robots similares en un espacio limitado. La posibilidad de detectar compañeros y rivales permite diseñar una estrategia de juego mucho más elaborada y, por ende, tener un mejor desempeño a la hora de jugar. La principal dificultad reside en que los robots que se usan para este tipo de tareas cambian de forma y pueden aparecer con aspectos muy distintos durante el desarrollo de sus tareas.

En la presente tesis se desarrolla una alternativa de solución para detectar robots móviles, intentando obtener detecciones consistentes de distinto tipos de robots, a través de algoritmos estadísticos.

1.1. **Objetivos generales**

El objetivo de esta tesis es contribuir al conocimiento en el área de la detección y seguimiento visual de objetos móviles como robots a través del procesamiento de imágenes.

1.2. **Objetivos específicos**

Los objetivos específicos del trabajo serán:

- Conocer el estado del arte en la detección visual de robots (enfocándose al futbol robótico).
- Detectar robots móviles Aibo y Humanoides (tipo Hajime) en imágenes estáticas.
- Optimizar la detección de robots móviles y lograr el seguimiento en tiempo real de éstos en secuencias de video.
- Implementar la detección y seguimiento de Aibo dentro de un robot móvil Aibo, para condiciones de juego de fútbol robótico.

1.3. **Antecedentes y Metodología**

Actualmente, se cuenta con el software de entrenamiento y detección de objetos mediante Adaboost, creado por Rodrigo Verschae [21]. La efectividad de este algoritmo se ha probado de manera muy satisfactoria con la detección de caras en imágenes. Además, tiene un respaldo teórico que ha sido estudiado profundamente en artículos de investigación científica [7-14, 16, 17].

El algoritmo de Adaboost es de tipo estadístico y requiere de una gran base de datos con gran variabilidad para su entrenamiento (en este caso imágenes de los objetos a detectar). El inicio de la investigación consiste en generar esta base de datos, luego entrenar el algoritmo y después probar su capacidad de detección para este tipo de objetos. Estos detectores serán probados en un par de aplicaciones distintas para comprobar su funcionamiento y utilidad. Para la evaluación práctica de los algoritmos se generan también bases de datos de prueba ("*benchmark*"), de manera a medir el rendimiento y tener una referencia para investigaciones futuras en el tema.

Una vez que la detección esté funcionando correctamente, se optimizará el algoritmo para que pueda funcionar en tiempo real intentando mantener el mismo rendimiento obtenido anteriormente. Luego se implementará este algoritmo dentro de los robots Aibo para que puedan detectar sus similares y se seguirá probando su velocidad y rendimiento en condiciones mas exigentes (como lo son las condiciones de juego y correr el algoritmo en paralelo con otras tareas). Por otro lado también se prueba un algoritmo de seguimiento de objetos que usa las detecciones como punto de partida y que tiene la ventaja de ser mas rápido.

Para realizar el trabajo de detección de robots móviles en imágenes y secuencias de video se trabaja en lenguaje C y C++, tanto para procesar las imágenes como para aplicar los algoritmos de detección, principalmente por su versatilidad y eficiencia. Además, al ser enteramente programado en C y C++ permite ser implementado en casi cualquier tipo de plataforma.

1.4. **Estructura del documento**

El presente documento consta de 5 capítulos distintos, los cuales se detallan a continuación.

El primer capítulo es de introducción y presenta una visión general del problema a resolver. Además, explica el interés del proyecto y señala los objetivos que se buscan lograr a través de este trabajo.

En el capítulo 2 se hace una revisión bibliográfica de trabajos sobre la detección de objetos, principalmente aquellos no-rígidos como lo son los robots móviles.

El tercer capítulo explica más profundamente el funcionamiento del algoritmo Adaboost para la detección de objetos, y describe algunos métodos que se han introducido para mejorar su rendimiento.

En el cuarto capítulo se explica la plataforma desarrollada, cómo se plantea el esquema de detección y cómo se crean las bases de datos y los clasificadores.

El quinto capítulo muestra los experimentos realizados para probar la plataforma desarrollada. También expone los resultados obtenidos y comenta su eficiencia y/o utilidad.

El sexto y último capítulo habla de las conclusiones del trabajo. Ahí se comentan si se los objetivos alcanzados, se resumen los resultados obtenidos y se señalan los posibles trabajos que se pueden realizar a futuro con lo que se logró en el desarrollo de este trabajo.

2. **Revisión Bibliográfica**

En esta sección se realiza un estudio de trabajos dedicados a la detección de robots móviles en imágenes enfocándose en la investigación que se ha hecho en torno a la Robocup (competencia enfocada a promover la investigación en temas relacionados con la inteligencia artificial y la robótica a trabes del futbol robótico) [23]. Primero se describen los métodos más usados, algunas otras alternativas, y luego se explicará la nueva estrategia que se ocupó para este trabajo.

2.1. **Esquemas clásicos de detección de robots**

En el ámbito de la Robocup, el método más usado para la detección de robots móviles está basado en el análisis de la distribución de colores en la imagen, aprovechando que el entorno es controlado y los objetos con los que se interactúa tienen colores específicos (por ejemplo en la *four-legged-league*, los equipos se distinguen con el color Rojo para uno y Azul para el otro). En la mayoría de los casos también se hace uso de líneas de escaneo, que consisten en recorrer píxel a píxel la imagen en cierta dirección para ver la transición de colores.

El equipo German Team que participa en la *four-legged league* de la Robocup [2, 15] hace uso de una estrategia compleja de búsqueda de robots Aibo a través de líneas de escaneo. Tienen un conjunto básico de líneas que sirven para la detección de todos los objetos en la cancha. Sin embargo, cuando encuentra una línea que contenga 3 píxeles del color de un robot (rojo o azul) se activa la detección. A partir de estas zonas de color se hace una nueva línea de escaneo que busca transiciones del tipo blanco-rojo-blanco o rojo-blanco-rojo (o con azul si se detectó ese color). Para cada transición de rojo a blanco (o azul a blanco), se generan 3 nuevas líneas de escaneo, una hacia abajo y dos otras a 45 grados a la izquierda y a la derecha hacia el piso de manera a tratar de encontrar el punto más bajo del robot (y con éste poder determinar su posición). Este método funciona correctamente para robots estáticos que están entre 20 y 150 cm., y es sumamente rápido

debido al bajo costo computacional de las líneas de escaneo. Sin embargo, es muy dependiente de la calibración de colores la cual, además de depender mucho de la luminosidad ambiente, tiende a empeorar cuando hay mucho movimiento en la imagen. Otra desventaja es que funciona sólo para robots de tipo Aibo.

Otro equipo de la four-legged league, que hace algo similar basado en color es el NUbot [3]. Este equipo también se enfoca únicamente en la detección de robots Aibo y busca manchas (blobs) de los colores posibles del robot, generando una detección a partir de la relación entre tamaños y distancias de las manchas de interés. Por otra parte hacen una distinción entre el rojo y el azul, ya que es mucho más difícil de detectar correctamente que el rojo (por las limitaciones de la cámara y porque el rojo es mas intenso), sobre todo en situaciones de juego. Para el azul, la generación de manchas se mejora a través de un detector de bordes que se combina con las manchas que aparecen en primera instancia. Luego, con las nuevas manchas mejoradas, se hace la relación entre éstas para generar la detección. Además de la detección, se intenta hacer una clasificación de la orientación del robot, también basada en las relaciones entre manchas de colores. Este algoritmo tiene las mismas ventajas y desventajas que el anterior, ya que depende de manera importante de la segmentación de colores.

Un tercer método, elaborado por el equipo CMUDash de la Universidad de Carnegie Mellon [4], consiste en buscar columnas de píxeles que no sean verdes ni naranjos debajo del horizonte (para que estén en la cancha), y agrupar esas columnas en cajas que forman obstáculos. Si estos obstáculos tienen suficientes píxeles de color rojo o azul, se hace la detección del robot en la imagen. Este algoritmo no tiene tantos problemas con los Aibo de color azul, pero sigue dependiendo fuertemente de la calibración de colores.

2.2. **Esquemas alternativos de detección de robots**

Otro enfoque utilizado para la detección de robots es aquel basado en la búsqueda y descripción de puntos característicos (llamados SIFT: *Scale Invariant Feature*

Transform). En el trabajo realizado por Loncomilla y Ruiz-del-Solar [1, 5], se ocupa este método para determinar la pose y orientación de la cabeza de los robots Aibo, además de identificar el número que lleva el Aibo en su cabeza (que se usa para distinguirlos durante un partido). La metodología consiste en calcular descriptores locales que sean invariantes a la escala y a la orientación, y luego comparar estos descriptores con una base de datos de cabezas de manera de determinar si en la imagen existen cabezas que tengan poses similares a las que se encuentran en la base de datos. Con esto, se encuentra un robot y además se determina directamente su identidad. La principal ventaja de este tipo de método es que tiene una buena tasa de detección y permite identificar directamente los robots. Además, con una combinación con detección de colores, se puede determinar el equipo del jugador (como se hace en [5]). Sin embargo, este método es muy lento para ser ocupado en paralelo con procesos de segmentación de colores y otro tipo de detección de objetos (pelotas, arcos, etc…).

La segmentación de colores es una etapa obligatoria en las actividades relacionadas con el fútbol robótico de la Robocup, ya que todos los elementos que se ocupan (cancha, pelota, arcos, robots, etc…), tienen colores específicos. Además, debido a la calidad de la cámara y a la capacidad computacional de los robots, es lo más indicado para tener un código que funcione en tiempo real. Es por esta razón que todos los equipos aprovechan la segmentación de colores para la detección de robots. Sin embargo, este enfoque es limitado ya que no toma en cuenta las deformaciones que puede sufrir el robot al moverse durante un partido, además de depender fuertemente de la luminosidad ambiente y la correcta segmentación de colores.

Los algoritmos basados en SIFT son una primera aproximación para la detección de robots no basada en colores, pero -además de ser lentos-, no permiten detectar robots completos sino que sólo la cabeza.

Para esta tesis se decidió hacer una detección de robots que no estuviese basada en la segmentación de colores (debido a las limitaciones antes mencionadas). Debido a los

excelentes resultados de la detección de otro tipo de objetos (principalmente caras), obtenidos por lo métodos basados en Adaboost [8, 10, 11, 16], se decidió probar la implementación de un detector de robots basado en este algoritmo. La versión que se implementó se basa en Adaboost, pero también tiene otras características importantes en su estructura, como la realimentación (*bootstrapping*) y las cascadas de clasificadores, entre otras cosas. A continuación, se hace una descripción de la teoría detrás del método ocupado, y de cómo funciona en la práctica.

3. **Clasificadores de tipo boosted con cascadas**

El método de detección que se decide utilizar se basa en la imagen, es decir en las propiedades de los píxeles (en este caso la intensidad). Fue implementado por Rodrigo Verschae basado en el algoritmo de Adaboost que apareció en 1995 publicado por Freund y Schapire [17]. Otro método de detección de objetos usado comúnmente es aquel basado en características, mediante el cual se busca encontrar un modelo de las características del objeto a encontrar. Sin embargo, estos métodos suelen requerir modelos complejos difíciles de caracterizar y sintonizar.

3.1. **Adaboost**

El nombre Adaboost viene de la mezcla entre Adaptive y Boosting, que son las dos características principales que ocupa el algoritmo. Los algoritmos de Boosting buscan encontrar un clasificador general o "fuerte", en base a la combinación de una serie de clasificadores específicos o débiles (en general son reglas muy simples). Se dice "Adaptive" (o Adaptivo), porque el clasificador "fuerte" se va adaptando y seleccionando los clasificadores débiles en función de los resultados que va obteniendo durante el entrenamiento. Este es un clasificador de tipo estadístico ya que usa características que se basan en la información cuantitativa de uno o varios elementos de las imágenes a estudiar.

3.2. **Estructura del clasificador**

La estructura básica de un clasificador fuerte de tipo Adaboost es la siguiente:

$$H(x) = \sum_{t=1}^{T} \alpha_t \cdot h_t(x) - b$$

Donde $h_t(x)$ es un clasificador débil, α_t un factor que combina linealmente los clasificadores débiles y *b* representa un umbral de operación del clasificador. La respuesta

final del clasificador se toma como $Signo(H(x))$, donde un valor mayor que 0 representa una detección positiva del objeto y un valor menor o igual a 0 representa que no se encuentra el objeto en la ventana analizada.

Los clasificadores débiles son usados sobre características (features), calculadas en cada imagen (o patrón) procesado, y cada clasificador débil tiene asociado una sola característica. El diseño de estos clasificadores débiles se hace bajo el paradigma de la "partición de dominio con hipótesis débiles" [7]. Bajo este paradigma los clasificadores débiles hacen sus predicciones basadas en la partición del dominio. Estos hacen sus predicciones basados en la división de un dominio *X* tal que cada clasificador tenga un valor asociado a una partición de ese dominio. El dominio es divido en bloques disjuntos $X_1,...,X_n$, que cubren *X* por completo, y para los cuales $h(x)=h(x')$ para todos aquellos $x,x' \in X_j$. Con esto las predicciones de los clasificadores débiles dependen solamente del bloque en que cae la característica analizada. Como el dominio en el que se está trabajando es el de las características (o *features*), se define el dominio *F*, dividido en bloques disjuntos $F_1,...,F_n$, y un clasificador débil *h* tendrá una salida para cada bloque de la partición asociada a la característica que le corresponde de la forma: $h(f(x))=c_j \ni f(x) \in F_j$.

Para cada clasificador, el valor asociado a cada bloque de la partición (c_j, es decir su salida), es calculado de manera de minimizar una función que representa un límite del error de entrenamiento [7]. Este valor depende de los siguientes factores: (i) el número de veces que la característica cae dentro de este bloque de la partición (histograma), (ii) de la clase de los ejemplos (llamados y_i, en general son 2 clases, positiva y negativa), y (iii) de su relevancia o peso, denominado *D(i).* El valor c_j está dado por la siguiente fórmula:

$$c_j = \frac{1}{2}\ln\left(\frac{W_{+1}^{j}+\varepsilon}{W_{-1}^{j}+\varepsilon}\right) \quad \text{con} \quad W_l^{j} = \sum_{i:f(x_i)\in F_j \wedge y_i = l} D(i) = \Pr\left[f(x_i)\in F_j \wedge y_i = l\right] \text{ donde } l = \pm 1$$

donde ε es una parámetro de regulación.

Estas salidas c_j, calculadas durante el entrenamiento del clasificador, son guardadas dentro de una *'look up table'* (LUT) para aumentar la velocidad de evaluación. De esta forma, cada clasificador está definido por $h(f(x)) = h_{LUT}[x] = LUT[index(f(x)]$, siendo $index$ una función que retorna el índice (o bloque) asociado al valor de la característica $f(x)$ en la LUT. Esta implementación permite calcular la salida de los clasificadores débiles en tiempo constante, mientras que el tiempo de entrenamiento sólo aumenta linealmente con la cantidad de ejemplos de entrenamiento.

3.3. **Tipo de Características**

Las características que se usan para la detección de robots móviles son las llamadas Características Rectangulares ("Rectangular Features"), similares a los *wavelets* de Haar [22]. Éstas se calculan como la diferencia entre la suma de intensidad de los píxeles contenidos en distintas áreas rectangulares de la imagen. Las operaciones de restas se pueden calcular de manera muy rápida e independiente del tamaño de los rectángulos, gracias a la "imagen integral" desarrollada por Viola & Jones en otro paper sobre Adaboost [12]. La figura a continuación muestra una representación de estas características:

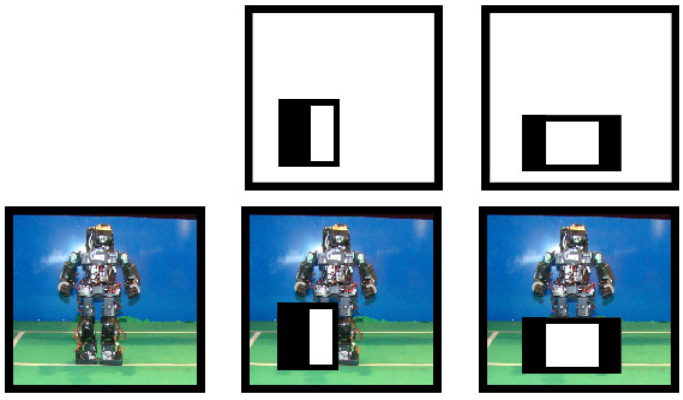


Figura 1: Características Rectangulares

Aparte de este tipo de características calculadazas sobre imágenes, existen cierto tipo de filtros y otros operadores. Sin embargo, para el caso de robots móviles, las características rectangulares obtienen mejores resultados.

3.4. **Cascadas de Clasificadores**

Para mejorar el rendimiento del sistema de detección, se pueden crear distintas configuraciones de combinación de clasificadores débiles y fuertes. En este caso se ocupó una configuración llamada Cascada Anidada de Clasificadores Fuertes (*'Nested Cascade of Boosted Classifiers'*). Esta configuración consiste en una serie de etapas de clasificación integradas, donde cada capa contiene un clasificador fuerte (del tipo "Adaboost"). La cascada completa funciona como un clasificador único que integra los clasificadores de cada etapa. La figura 2 muestra la estructura de este tipo de cascadas.

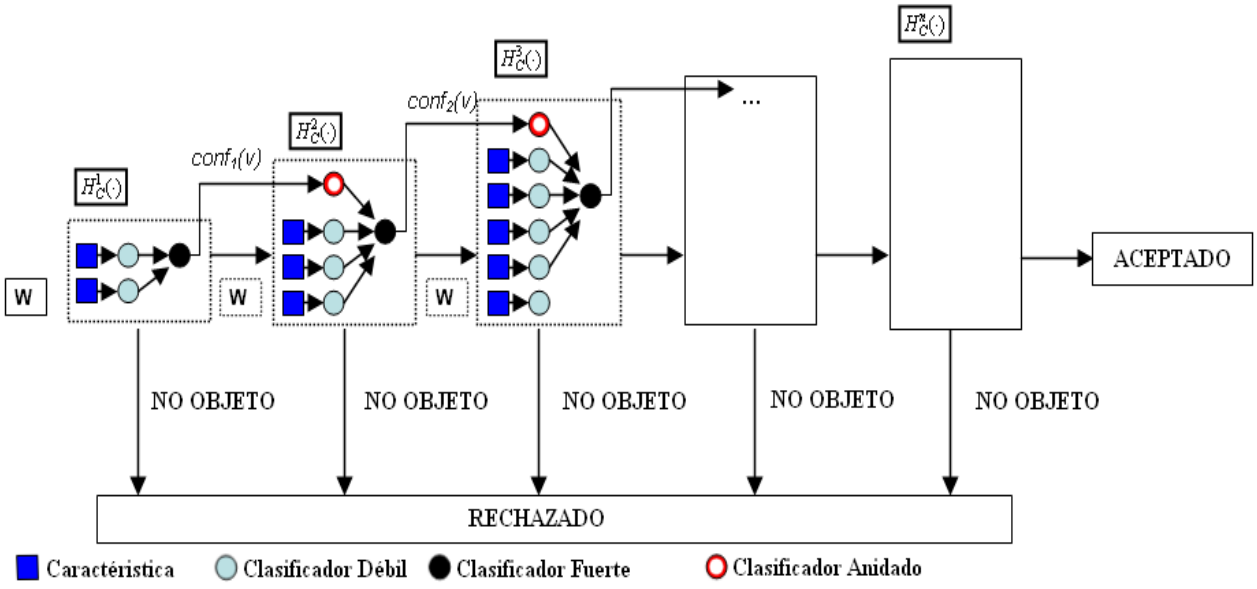


Figura 2: Estructura de una cascada anidada de clasificadores fuertes

Este tipo de cascadas se puede definir matemáticamente de la siguiente forma: una cascada anidada C, compuesta de M etapas, se define como la unión de M clasificadores fuertes H_C^k: $C = \bigcup_{k=1}^{M} \{H_C^k\}$.

Cada clasificador corresponde a la combinación (o anidamiento) de los clasificadores de las etapas anteriores. El cálculo de H_C^k hace uso del valor de los clasificadores anteriores de la siguiente forma:

$$H_C^k(x) = H_C^{k-1}(x) + \sum_{t=1}^{T_k} h_t^k(x) - b_k \quad .$$

La etapa 0 se inicializa con valor nulo: $H_C^0(x)=0$. Como se aprecia anteriormente $h_t^k(x)$ representan los clasificadores débiles, T_k el número de éstos en la etapa *k,* y b_k como un valor umbral para la clasificación.

Debido a la configuración anidada de C, su salida está dada por:

$$O_C(x)=\begin{cases}\text{sign}(H_C^q(x)) & H_C^k(x)\geq 0, k=1,...,q \wedge H_C^{q+1}(x)<0 \\ \text{sign}(H_C^1(x)) & H_C^1(x)<0\end{cases}$$

Con un valor de confidencia de una detección positiva dada por:

$$conf_C(x)=H_C^q(x) \ni H_C^k(x)\geq 0, k=1,...,q \wedge H_C^{q+1}(x)<0$$

Una de las principales ventajas de este tipo de estructura de clasificación es poder procesar rápidamente las ventanas que no se parecen al objeto de interés, mientras se procesan más lentamente aquellas que sí se parecen, ya que aquellas ventanas que no son aceptadas por las primeras etapas de las cascadas no siguen siendo procesadas por el resto de las etapas.

3.5. **Entrenamiento del clasificador**

Para cada objeto que se desea detectar (caras, robots, etc…), se necesita entrenar el clasificador fuerte, lo cual consiste en encontrar los clasificadores débiles que lo componen y cómo se ordenan entre ellos. Para esto se requiere tener una base de datos de entrenamiento que consiste en un grupo de imágenes que contengan el objeto a detectar, y otra que no lo contenga. Estas bases de datos deben ser construidas manualmente por una persona, y se hace en general a través de recopilación de imágenes y/o la anotación de secuencias de video. En general, esta etapa puede tomar mucho tiempo ya que se requiere una gran cantidad de imágenes que además deben ser representativas del objeto a detectar (deben contener variabilidad de posición, iluminación, contraste, etc…).

De acuerdo a la sección anterior, se puede entrenar un clasificador fuerte como una combinación lineal de clasificadores débiles, o también se puede entrenar una cascada de clasificadores que se genera al poner en serie un grupo de clasificadores fuertes. A

continuación se muestra y explica un pseudo-código para ambos casos. El entrenamiento de un clasificador fuerte mediante la combinación de clasificadores débiles se puede resumir de la siguiente forma (basado en el algoritmo de Viola & Jones [12]):

- Dado un conjunto de imágenes (x_1, y_1) ,…, (x_n, y_n) donde $y_i = +1, -1$ para ejemplos positivos y negativos respectivamente.
- Se inicializan los pesos $w_i(1)(i = 1,..., n)$ de manera que estos esten normalizados: $\sum_{i=1}^{n} w_{i,}(1) = 1$.
- *'For'* t = 1, …, T:
    1. Sea ε_j la suma de los errores de clasificación para el clasificador débil $h_j : \varepsilon_j = \sum_{i=1}^{n} w_i(t) I[y_i \neq h_j(x_i)]$.
    Donde
    $$I[\gamma] = \begin{cases} 1, si\ \gamma = Verdadero \\ 0, si\ \gamma = Falso \end{cases}$$
    Se elije (dentro de los clasificadores generados), el clasificador h_t que minimiza la suma de los errores de clasificación
    $$h(t) = \arg \min_{h_j \in H} \varepsilon_j .$$
    2. Calcular la suma de los errores de clasificación $\varepsilon(t)$ para el clasificador $h(t)$ elegido.
    3. Se calcula
    $$\alpha(t) = \frac{1}{2} \log\left(\frac{1 - \varepsilon(t)}{\varepsilon(t)}\right).$$
    4. Se actualizan los pesos:
    $$w_i(t+1) = \frac{w_i(t) \exp(-\alpha(t) y_i h(t)(x_i))}{Z(t)}$$
    Con Z(t) el factor de normalización siguiente:
    $$Z(t) = \sum_{k=1}^{n} w_k(t) \exp(-\alpha(t) y_i h(t)(x_k))$$
- Finalmente, el clasificador fuerte definitivo queda como:
$$H(x) = signo\left(\sum_{t=1}^{T} \alpha(t) h(t)(x)\right)$$

Cuadro 1: Pseudo-código del algoritmo de entrenamiento de un clasificador fuerte mediante la combinación lineal de clasificadores débiles.

Lo más complicado para formar un clasificador fuerte es la selección de los clasificadores débiles que lo componen, y cómo se combinan entre ellos. El pseudo-código del Cuadro 1 muestra como cada clasificador débil es elegido (aquel que tiene menor error de clasificación dentro del conjunto de imágenes de entrenamiento), y tiene asignado su propio peso que está directamente relacionado con su error respectivo. Además, los pesos permiten que los primeros clasificadores tengan más importancia en el resultado final. Generalmente los primeros clasificadores seleccionados tienen un error entre 0.1 y 0.3, mientras que los últimos se sitúan entre 0.4 y 0.5. Como se explica anteriormente, para este trabajo se usan principalmente características rectangulares.

En el caso anterior, una vez que se tiene el clasificador fuerte, cada ventana de 24 por 24 píxeles (tomadas de la imagen a procesar) que se quiera clasificar, tiene que ser medida con las T características débiles del clasificador. Dado que en general un clasificador tiene alrededor de 200 características, y que en una imagen relativamente pequeña de 120 por 120 píxeles se pueden procesar hasta alrededor de 10000 ventanas de 24 por 24, el procesamiento de clasificación puede tardar bastante. El tamaño de la ventana puede ser distinto (desde 20 por 20 hasta 48 por 48), sin embargo se eligió un cuadrado de 24 pixeles porque es el que entrega mejores resultados de detección en general.

Es por esta razón que se han creado configuraciones más eficientes para formar clasificadores fuertes, de manera de reducir los cómputos promedios necesarios para clasificar una imagen (hay que recordar que esta aplicación busca trabajar en tiempo real).

Como se ve anteriormente, la Cascada Anidada de Clasificadores Fuertes cumple con estos requisitos, ya que en vez de que cada ventana a analizar tenga que pasar por todos los clasificadores débiles, éstas tienen que pasar por pequeñas etapas de clasificadores fuertes que pueden rechazarla, o dejarla pasar a la siguiente etapa. De esta forma, muchas ventanas son rechazadas en la primera cascada, mientras que pocas pasan hasta el final, con lo cual se reduce considerablemente el procesamiento requerido para clasificar una

imagen. Esta estructura permite, además, mantener -e incluso mejorar-, las tasas de detecciones de los clasificadores.

- Se seleccionan valores para f , la máxima tasa de falsos positivos por etapa y d la tasa mínima de detecciones correctas por cada etapa.
- Se selecciona la tasa global de falsos positivos que se quiere alcanzar F_{target} .
- P = conjunto de ejemplos Positivos
- N = conjunto de ejemplos Negativos
- $F_0 = 1.0; D_0 = 1.0$
- $i = 0$
- While $F_i > F_{\text{target}}$
  - $i \leftarrow i + 1$
  - $n_i = 0; F_i = F_{i-1}$
  - While $F_i > f * F_{i-1}$
    - $n_i \leftarrow n_i + 1$
    - Se usa P y N para entrenar un clasificador con n_i características usando Adaboost (pseudo-código en Cuadro n°1).
    - Evaluar la cascada actual en el conjunto de validación para determinar F_i y D_i .
    - Disminuir el umbral pare el $i-esimo$ clasificador para que la etapa actual de clasificación tenga una tasa de detección de al menos $d * D_{i-1}$ (esto también afecta a F_i).
- $N \leftarrow \emptyset$
- Si $F_i > F_{\text{target}}$ entonces se evalúa la cascada actual de clasificadores en el conjunto de imágenes negativas y se agregan las falsas detecciones al conjunto N.

Cuadro 2: Pseudo-código del algoritmo de entrenamiento de una Cascada Anidada de Clasificadores Fuertes.

Como la estructura del clasificador final es más compleja, ahora hay más parámetros que elegir, y éstos tienen una influencia importante en el resultado que se obtendrá al final del entrenamiento. Además, en este caso se usan 2 conjuntos adicionales de imágenes para ir midiendo la tasa de detección del algoritmo. Estos conjuntos son llamados conjuntos de

validación positivo y negativo. El pseudo-código recién mostrado (cuadro 2) ejemplifica cómo se realiza el entrenamiento de una Cascada Anidada de Clasificadores Fuertes.

Como se puede ver, el proceso de entrenamiento es bastante más complejo y más lento que en el caso anterior pero, dado que se tiene que hacer una sola vez para cada detector que se desea entrenar, este problema no es tan relevante (veremos, sin embargo, que igual requiere hacer ciertas reducciones de procesamiento pues el entrenamiento puede llegar a durar semanas). Los clasificadores fuertes que forman las cascadas tienen ahora una cantidad más reducida de clasificadores débiles (en general desde 7 hasta 15).

Este tipo de cascadas de clasificadores obtienen velocidades del orden de 10 veces mayores a las del caso anterior, razón por la cual son las más usadas usualmente para detectar objetos. Por esta razón, y por su buen desempeño como detector, se usan este tipo de cascadas en esta tesis.

Los conjuntos de imágenes de entrenamiento son muy importantes para obtener un buen detector final, ya que es a partir de éstos que se decide la estructura final de los clasificadores. Es fundamental que en el conjunto de imágenes exista la máxima variabilidad posible, principalmente en términos de iluminación y contraste. Para aliviar un poco la necesidad de bases de datos muy grandes (por ende el tiempo que demora generarlas), y para mejorar y hacer más rápido el entrenamiento, se han creado distintos métodos [19], uno de los cuales se describe a continuación.

3.6. **Procedimiento de "Bootstrapping"**

En teoría, cualquier ventana en una imagen mayor a 24 por 24 píxeles, que no contenga el objeto a detectar, sirve como imagen para el conjunto de ejemplos negativos. Claramente, tratar de incluir todas estas ventanas a partir de las imágenes de la base de datos no es factible. Para reducir la selección, y mejorar la frontera entre objeto y no-objeto, se deberían elegir ventanas que se parezcan al objeto a detectar, pero que no lo contengan (similitud de tamaño, forma, etc…). Esto se resuelve usando el procedimiento de

“bootstrapping” (algo como una realimentación), que consiste en ir entrenando el clasificador iterativamente, aumentando cada vez el conjunto de imágenes negativas con imágenes que no contengan el objeto, pero que hayan sido mal clasificadas por el detector que se está entrenando.

Cuando se entrena una cascada de clasificadores, es útil realizar el “bootstrapping” en dos situaciones: una vez antes de entrenar una nueva etapa de la cascada (bootstrap externo), y varias veces durante el entrenamiento interno de cada etapa de la cascada (bootstrap interno).

De esta forma se aumenta la base de datos automáticamente, y además se van reforzando la frontera entre objetos y no-objetos para que el clasificador no se confunda con cosas que tengan forma y tamaño similar a lo que se quiere detectar.

4. **Plataforma de Detección Desarrollada**

Como se vio anteriormente, para la detección de robots en imágenes se decide usar Adaboost. En este capítulo se explica la metodología usada para implementar este algoritmo, al igual que el esquema de detección.

4.1. **Esquema de detección**

En la siguiente figura se puede ver el esquema global utilizado para realizar la detección de robots (Aibo en este caso).

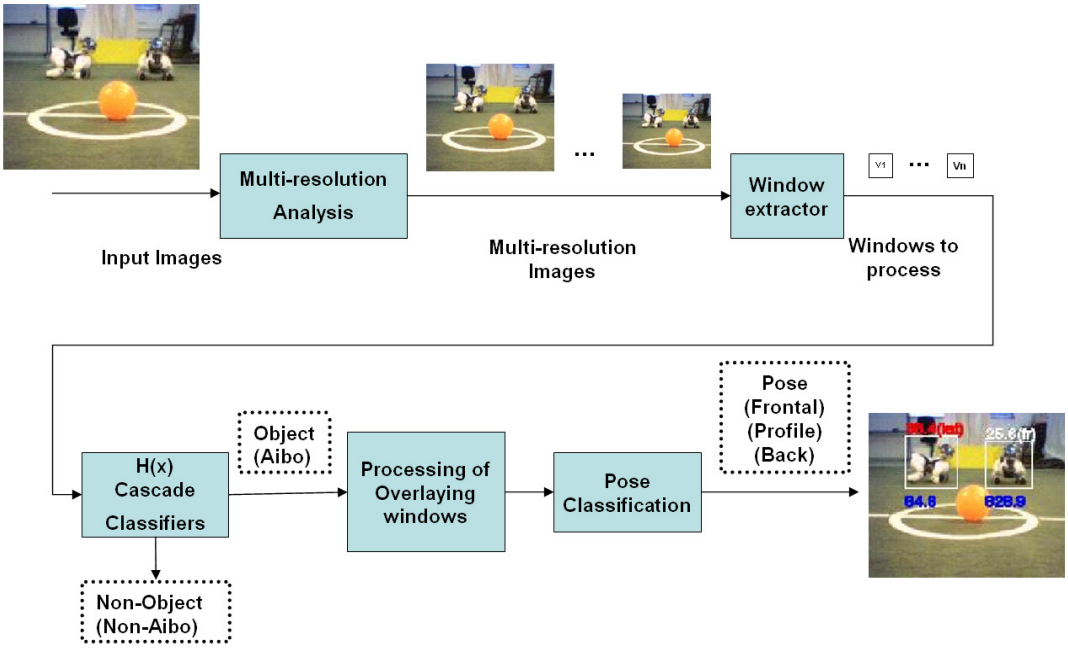


Figura 3: Esquema de detección

La entrada que recibe el algoritmo es una imagen cualquiera, ya sea de un video o de una selección de imágenes. Lo primero que se hace es analizar la imagen en multi-resolución, ya que esto permite detectar robots a distintas escalas. En esta etapa, se extraen ventanas de 24 por 24 píxeles de la imagen en tamaño original, luego se reduce el tamaño de la imagen por un factor dado (de 1.2 en general), y se vuelven a extraer ventanas de 24 por

24. Esto se repite hasta que la imagen original queda del mismo tamaño de las ventanas que se extraen.

La siguiente etapa es analizar cada una de las ventanas extraídas. Esto se hace con el bloque llamado “Cascade Classifiers” (aquí es donde se aplica el concepto principal de Adaboost). En esta etapa, se decide cuáles ventanas pertenecen a la clase positiva (en este caso robot) y cuales no.

Luego, la ventana llamada “Processing of Overlaying windows” hace un análisis global de las detecciones positivas, y las fusiona para generar las detecciones de los robots en la imagen. Esta etapa entrega como salida la posición y el tamaño del robot detectado en la imagen. La fusión de las ventanas se hace sacando un promedio de las coordenadas de cada esquina que estén a cierta distancia entre ellas.

Finalmente, se puede agregar un bloque facultativo llamado “Clasificador de pose”, en el cual se analiza la pose de cada uno de los robots detectados (pose frontal, lateral o trasera). Este último bloque pasa los detectores de pose sobre las detecciones ya encontradas y elige la pose que tenga mayor confidencia (siempre que sea mayor a cierto umbral).

En resumen, en la entrada se tiene una imagen y en la salida se tiene la misma imagen con la información de la posición del robot en la imagen y su pose (si es que hay robot).

Este esquema tiene varios parámetros que se pueden modificar y permiten tener control sobre distintos aspectos del programa de detección. Dentro de éstos, uno de los principales es el factor que reduce la imagen original. El principal trade-off que aparece con este parámetro es que se puede tener un algoritmo muy rápido pero con baja tasa de detección, versus un algoritmo más lento con mejor tasa de detección.

El equilibrio entre estos dos factores depende principalmente de la aplicación para la cual se quiera usar el algoritmo. Si lo mas importante es que la aplicación sea en tiempo real, se necesita un algoritmo más rápido donde no importa tanto no detectar algunos casos, mientras que si lo que la aplicación necesita es una buena tasa de detección para funcionar, es mejor tener un algoritmo más lento pero con mejores detecciones.

Otro parámetro importante que se puede controlar es un umbral sobre la confidencia. Si se toman detecciones positivas como aquellas que tienen un alto nivel de confidencia se obtendrán pocos falsos positivos, pero bajara la tasa de detección. Por el contrario, si se toman detecciones a partir de un umbral mas bajo de confidencia, aumentara la tasa de detección pero también aumentara la cantidad de falsos positivos.

4.2. **Entrenamiento de los clasificadores**

4.2.1. **Generación de Base Datos de entrenamiento**

En la sección anterior se explica el esquema global de detección utilizado. En esa descripción se habló del bloque "Cascade Classifiers", en el cual se ejecutan principalmente Adaboost. Para que éste funcione bien, se necesita un clasificador entrenado para detectar los objetos de interés (tal como se explicó en el capítulo 2).

El Algoritmo de Adaboost busca encontrar ciertas características que se repitan continuamente en todas las imágenes del objeto a detectar, y eliminar aquellas que no forman parte de éste. Es por esta razón que para generar un buen clasificador se necesita una gran cantidad de imágenes que tengan, además, distintas condiciones de entorno e iluminación.
Para entrenar el clasificador, es necesario generar una base de datos grande y diversificada, de tal forma que el objeto a detectar pueda ser caracterizado lo más globalmente posible. De esta forma el clasificador puede sacar más información de todo lo que se repite en el objeto y no de elementos particulares que no representan a éste. La idea es tener un conjunto que sea lo más representativo posible del objeto a detectar.

La construcción de la base de datos (tanto para los Aibo como para los Humanoides), se hizo sacando varios videos del objeto a detectar con distintas características de luminosidad, fondo y pose. Para los Aibo se usaron, además, imágenes sacadas por la cámara de los Aibo, mientras que para los Humanoides se ocuparon videos publicados en internet de distintos tipos de Humanoides [18]. Con esto, el detector es independiente de

la luminosidad y de los distintos fondos que pueda tener la imagen donde se quiere detectar el robot.

Una vez obtenidos los videos, éstos son divididos en imágenes. En cada imagen se marcan 4 puntos extremos del robot (el punto mas a la izquierda, más abajo, más arriba y más a la derecha), y luego se recorta un cuadrado centrado en el robot de arista igual al máximo entre el ancho y el alto marcado por los puntos. En el diagrama a continuación se ve más claramente el proceso realizado:

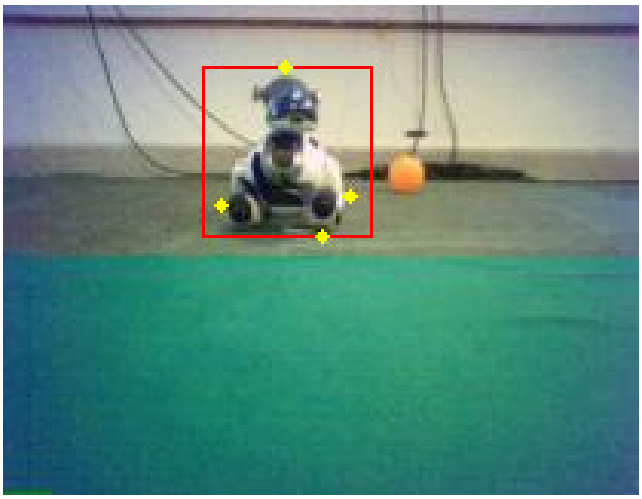


Figura 4: Diagrama de recorte de un Aibo para la base de dato de entrenamiento

Una vez recortado el cuadrado, éste se reduce a un tamaño de 24 por 24 píxeles y se convierte a blanco y negro. Esta última imagen es la que va a ser usada directamente para el entrenamiento del clasificador. En la figura 5, a continuación, se puede ver claramente todo el proceso necesario para generar las imágenes de entrenamiento.

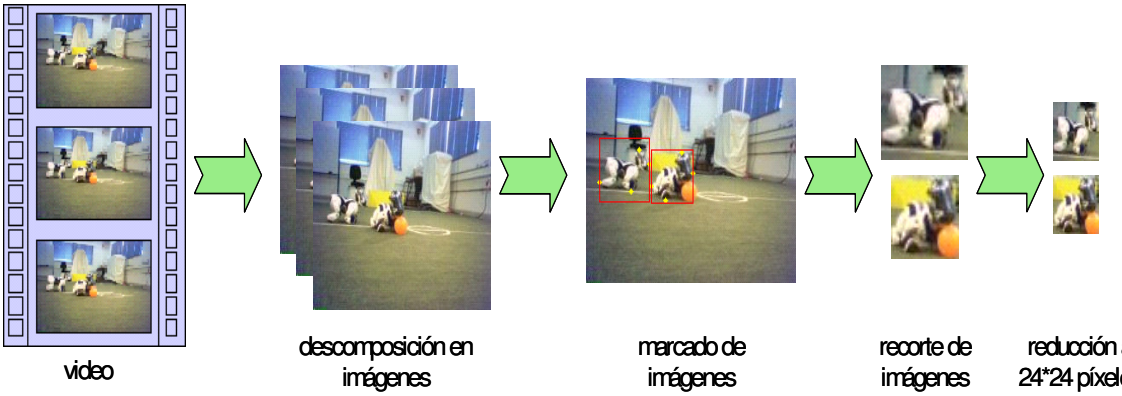
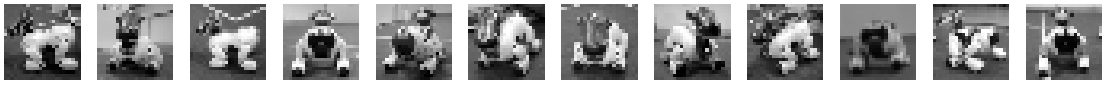
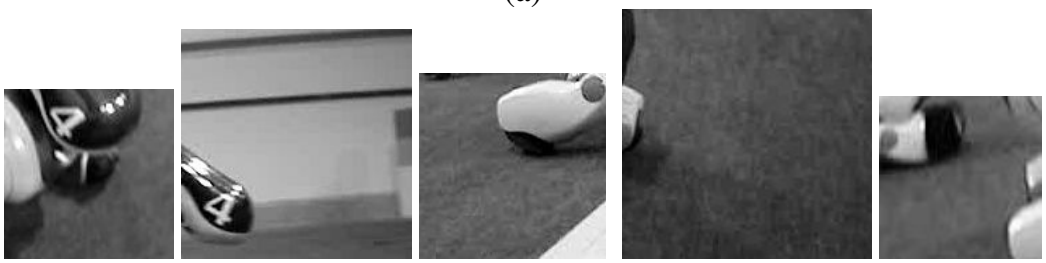


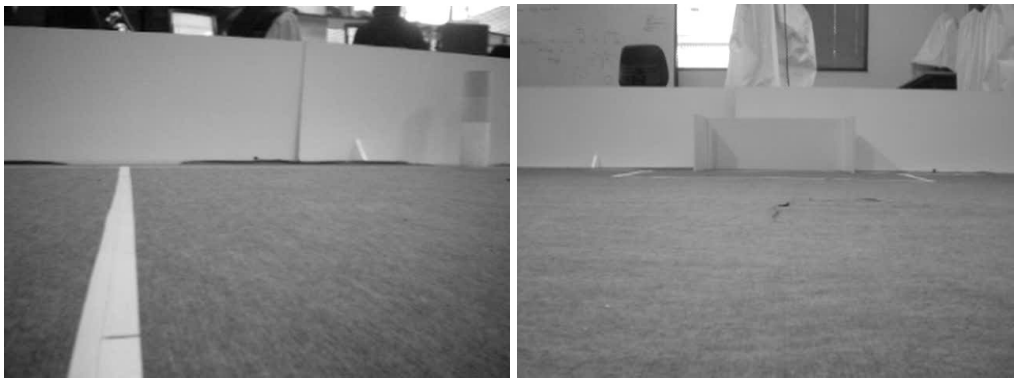
Figura 5: Diagrama del proceso de creación de imágenes de entrenamiento

Aparte de ésta, también se generó una base de datos de elementos “negativos”, es decir, que no tuvieran ningún robot. Para esta base de datos, sólo se necesitan imágenes de cualquier tamaño que no contengan Robots, o que sólo contengan una pequeña parte de Robot. A continuación se muestra una lista de ejemplos de casos positivos y de casos negativos. Hay que notar que los ejemplos positivos son todos del mismo tamaño (24 por 24 píxeles), mientras que los ejemplos negativos pueden tener cualquier tamaño (mientras sea superior a los ejemplos positivos). El mismo algoritmo de entrenamiento saca las ventanas necesarias sabiendo que no va a haber nunca un robot ahí.



(a)





(b)

Figura 6: Imágenes de la base de dato de entrenamiento:
(a) Ejemplos positivos – (b) Ejemplos negativos

Este proceso de entrenamiento se repite 3 veces para los Aibo y una vez para los robots Humanoides. En el caso de los Aibo, se generaron 3 clasificadores: un clasificador de Aibo en pose frontal, otro en pose lateral y el último en pose trasera. A continuación, se muestra una tabla que resume la cantidad de imágenes usadas para entrenar cada uno de los clasificadores.

Clase	# Ejemplos Positivos (Entrenamiento)	# Ejemplos Positivos (Validación)	# Imágenes Negativas (Entrenamiento)	# Imágenes Negativas (Validación)
Aibo Frontales	3115	3115	5946	2550
Aibo Laterales	4263	3624	5946	2550
Aibo Traseros	1528	1528	5958	2562
Humanoides	3506	3500	5958	2562

Tabla 1: Resumen de la cantidad de imágenes usadas para los clasificadores

4.2.2. **Generación de los detectores**

Una vez que se tienen las bases de datos de imágenes se puede proceder al entrenamiento de los detectores, tal como se vio en el capítulo 2. Este proceso es muy importante ya que de aquí salen los detectores definitivos que se van a usar.

El algoritmo que se usa para entrenar los detectores toma entre 2 y 5 días en promedio, corriendo a tiempo completo en un servidor, para ir generando la cascada de clasificadores. Sin embargo, debido a la complejidad del entrenamiento y a los distintos parámetros que influyen directamente en el rendimiento del clasificador, se necesitan varias iteraciones del proceso de entrenamiento de manera a obtener un detector final con el mejor rendimiento posible.

El siguiente archivo de configuración muestra la cantidad de parámetros que existen para realizar un entrenamiento:

```
# Identifier of the training:
#########################
id = "Traseros4"
#########################
# Training Sets
#########################
# Negative lists (for the bootstrapping):
negative_val_set_filelist ="NoPerro_Val_Set_3.txt"
negative_train_set_filelist ="NoPerro_Train_Set_3.txt"

# Positive examples:
positive_train_set_0 = "/home/marenas/NuevoTrain/Trainer/bin/Ent_Traseros1_24.bin"
positive_val_set_0 =  "/home/marenas/NuevoTrain/Trainer/bin/Val_Traseros1_24.bin"

N_CLASSES_OB = 1

#########################
# Training Parameters (read by Train_Cascade)
#########################
max_FPR_layer = 0.25 # in the validation dataset entre 0.2 y 0.5
min_DR_layer = 0.999 # in the validation dataset mínimo 0.995

#########################
# Training Parameters (read whitin train_Cascade_MO() at Adaboost-MO.c)
#########################
n_bootstrapping_steps = 5
n_bootstrapped_examples = 400 # 400 cuánto le agrego a cada conjunto nuevo
n_negative_examples = 2400 # 2400 conjunto inicial de etapa
n_negative_examples_eval = 2000 # 2000 para validacion (no agrega más fotos)
sampling_factor = 40.0 # porcentaje de features que considera
```

########################

 # Features

########################

nluts = 1 # Number of LUTs for the rectangular features

nbins = 24 # 128 # 72

En este archivo, todo lo que se encuentre detrás de un símbolo # representa un comentario, y los parámetros en si son los representados en los siguientes campos:

- El campo "*id*" identifica qué detector se está entrenando (en este caso es la cuarta iteración del detector de Aibo visto desde atrás).
- Los campos "*negative_val_set_filelist", "negative_train_set_filelist", "positive_train_set_0"* y "*positive_val_set_0",* representan los conjuntos de imágenes usados para el entrenamiento.
- Los campos "*max_FPR_layer"* y "*min_DR_layer",* son la máxima tasa de falsos positivos por etapa y la tasa mínima de detecciones correctas por etapa (los parámetros *f* y *d* del cuadro n°2).
- Los campos "*n_bootstrapping_steps", "n_bootstrapped_examples", "n_negative_examples",* y *"n_negative_examples_eval",* son parámetros usados para el "bootstraping" durante el entrenamiento.
- El *"sampling_factor"* representa el porcentaje de características que se van a usar durante el entrenamiento del total posible generable.
- Finalmente los campos "*nluts"* y *"nbins"* son parámetros de los features (en este caso rectangulares) que se van a usar.

Dentro de este conjunto de parámetros, los más importantes son el *f* y el *d.* Estos son los que más influencia tienen en el resultado final del clasificador y los que más varían entre las distintas iteraciones del entrenamiento de los detectores. Otros parámetros importantes son los del "bootstrapping", dado que -como se ve anteriormente-, el método puede ayudar considerablemente a mejorar las tasas de detecciones finales.

Para cada uno de los detectores finales creados se realizan entre 5 y 10 iteraciones de entrenamiento. Esta es una de las tareas que más tiempo demora en la generación de buenos detectores, debido al tiempo que demoran los entrenamientos. Además, es necesario realizar varias iteraciones, ya que la única forma de ver si un detector es mejor que otro es a través de pruebas de los detectores finales en condiciones reales de juego. Sin embargo, como el proceso se deja corriendo en servidores especialmente habilitados para eso, se pueden realizar otras tareas en paralelo al entrenamiento (comenzar a medir los primeros detectores, avanzar en las implantaciones del código en otras plataformas, etc…).

4.3. **Plataformas de Prueba**

Dado que el código del algoritmo ocupado está programado en el lenguaje C y C++, el esquema de detección se puede probar fácilmente en distintas plataformas. Para esta tesis se probó en distintos computadores (de escritorio y portátiles), y dentro de los robots Aibo (sus características están descritas más adelante). A futuro se piensa implementar, además, en robots Humanoides que usan PDA's (agendas personales electrónicas), para procesar la información, y también en los nuevos robots NAO que corren con linux.

5. **Experimentación y Resultados**

En esta sección se evalúan los resultados obtenidos para las distintas aplicaciones del método de detección desarrollado. Se ven las ventajas y desventajas de cada aplicación, como también la calidad de los resultados y los tiempos de procesamiento.

5.1. **Detección de robots en imágenes**

La primera aplicación consiste en hacer un estudio de la calidad del detector entrenado y probarlo en un rango global de situaciones. Para esto, se generan dos bases de datos de prueba con una serie de imágenes de juego tomadas desde un Aibo y otras de un video de un partido de Humanoides. Para los Aibo, se generan imágenes con distintas condiciones de luminosidad, fondos, cantidad de Aibo en la imagen, etc…, para poder estimar la tasa de detección que va a tener el algoritmo en cualquier tipo de condición de juego. En el caso de los Humanoides, se tomó un video que tuviera distintas situaciones de juego y una cantidad variable de robots en las imágenes. La siguiente tabla resume la información sobre las imágenes de las bases de dato:

Base de datos de evaluación	# Imágenes	# de Aibo (Frontales)	# de Aibo (Laterales)	# de Aibo (Traseros)	# Humanoides	Tamaño de la imagen
AIBODetUChileEval	724	344	489	180	-	208x160
HDetUChileEval	244	-	-	-	493	640x480

Tabla 2: Resumen de la cantidad de imágenes contenidas en la base de datos de evaluación.

5.2. **Resultados**

Para la evaluación del detector presentado, sobre las bases de datos de evaluación recién mencionadas, los resultados se presentan en forma de un gráfico de la Tasa de Detección (TD, es decir el porcentaje de robots correctamente detectados dentro del total) contra Falsos Positivos (FP, es decir la cantidad de ventanas incorrectamente marcadas como robot donde no se encontraba uno de estos) en la tabla 3 y la figura 7. Para la

clasificación de Pose, los resultados muestran el porcentaje de correcta clasificación de las Poses de los Aibo. Se muestran también imágenes de los resultados obtenidos para esta aplicación (figura 8). Se seleccionan distintos puntos de operación que dependen principalmente del umbral con el cual se acepta una detección como positiva. Mientras más bajo el umbral, más detecciones aparecen, pero también pueden aparecer más falsos positivos.

Detector / Objetivo	TD %	FP	TD %	FP	TD %	FP	TD %	FP	TD %	FP
Frontal /Aibo Frontales			89.4	254	84.4	57			74.5	18
Lateral/ Aibo Laterales	94.7	98	90.4	70			81.3	42		
Trasero /Aibo desde Atrás			89.9	166	85.6	76	79.8	27		
Frontal / Todos los Aibo			90.0	392			82.9	183	73.4	95
Múltiple / Todos los Aibo	94.8	392	88.6	183	84.3	114	80.1	52		
Humanoides	94.8	590	92.2	123					75.9	3

Tabla 3: Puntos de operación de los detectores implementados sobre las bases de datos de evaluación.

Para la base de datos de los Aibo, la primera prueba consiste en evaluar cada detector de forma independiente en la clase específica para la cual este detector estaba diseñado (Detector frontal evaluado sobre los Aibo de frente, etc…). En este caso, no se toman en cuenta los robots detectados que no correspondían a la pose evaluada (es decir un robot lateral encontrado por el detector de frente no es tomado en cuenta). El mejor resultado obtenido en esta etapa es el del detector de Aibo Laterales con una tasa de 90.7% de detección y 70 Falsos Positivos (FP) en 724 imágenes. Los detectores frontal y trasero obtuvieron una tasa de detección de alrededor de 85%, con cerca de 70 falsos positivos.

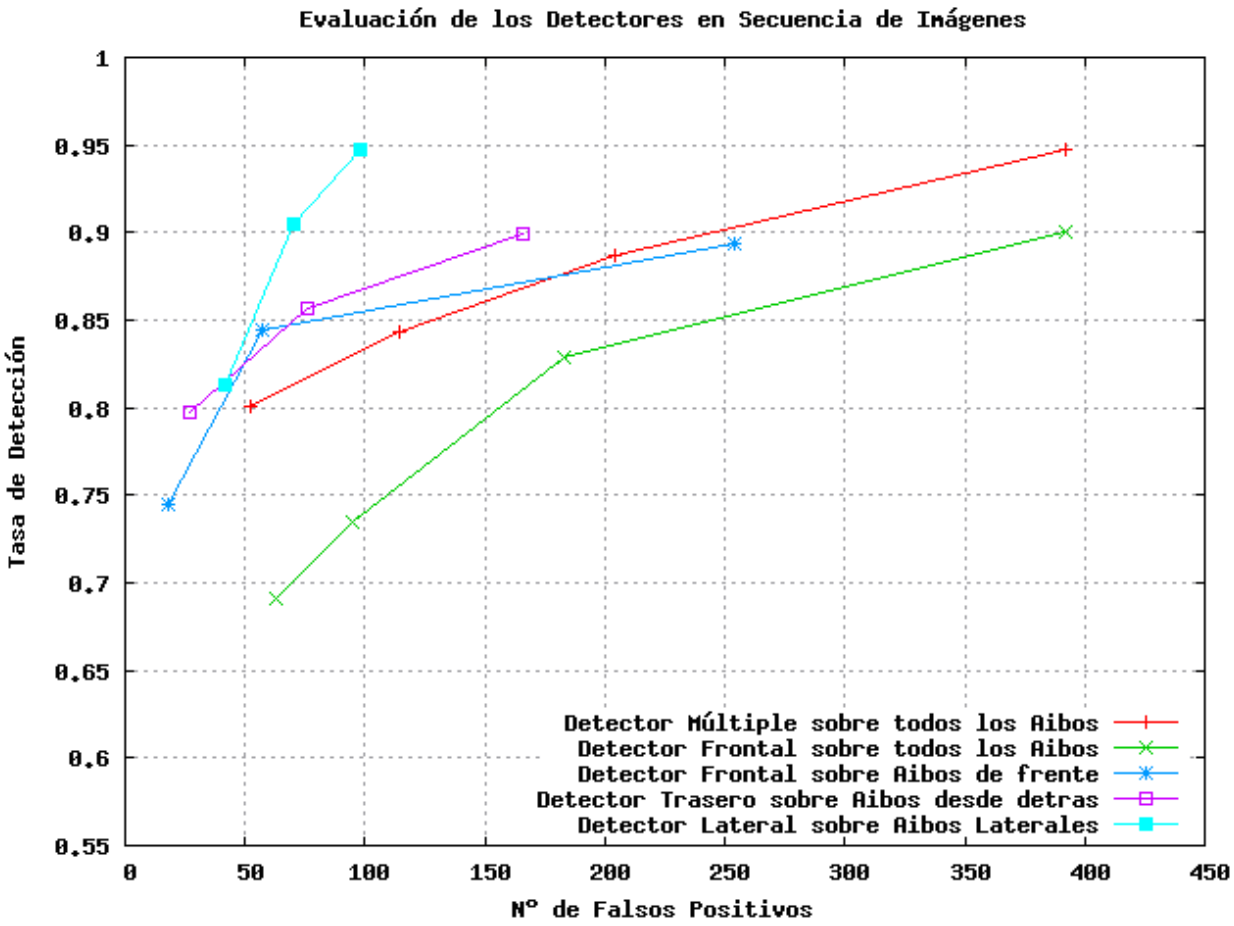


Figura 7: Curvas de detección en la base de datos de Aibo.

La segunda prueba que se realiza para evaluar los clasificadores, es calcular el desempeño de los detectores sobre todos los Aibo, independiente de su pose. Los detectores logran encontrar muchos de los Aibo detectados en otras poses, pese a que no fueron entrenados para eso (esto se debe probablemente a la relación y simetría de contrastes que tienen los Aibo). En este caso, el detector frontal es el que obtiene mejores resultados, logrando una tasa de detección de cerca de 90% con 392 Falsos Positivos.

La tercera prueba (Detector Múltiple sobre todos los Aibo en la figura 7), consiste en probar todos los detectores en paralelo sobre cada imagen. Dado que en algunos casos los Aibo son detectados por más de 1 detector, las detecciones sobrepuestas se fusionaban eligiendo aquella con mayor confianza. Para esta prueba se logra una tasa de detección de 94.8%, con 392 Falsos Positivos sobre las 724 imágenes de la base de datos. Al pasar 3

clasificadores distintos, se pueden encontrar más Aibo que pasando uno solo, sin aumentar notablemente la tasa de Falsos Positivos. Este proceso de detección, sin embargo, es alrededor de 3 veces más lento que los otros, por la misma razón.

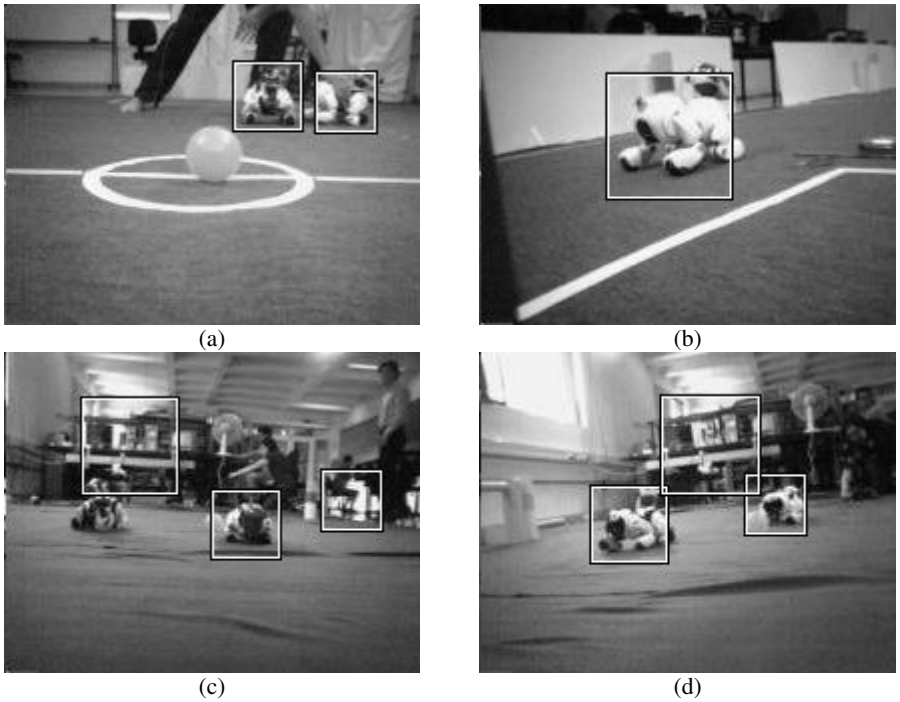


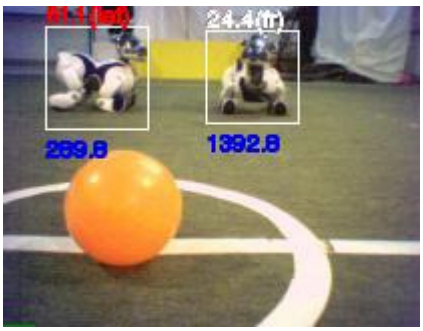
Figura 8: Resultados de detecciones sobre la base de datos de Evaluación de los Aibo (AIBODetUChileEval). En (a) y (b) se ven detecciones frontales, laterales y traseras. En (c) y (d) se ven 3 detecciones correctas, 3 falsos positivos y un Aibo no detectado.

La última prueba que se realiza para probar los clasificadores, es usarlos para estimar la pose de los robots en la imagen. Para esto, se usa el detector frontal como detector genérico (detectando todos los Aibo, con los mismos parámetros que obtuvieron un 90% de tasa de detección y 392 Falsos Positivos), para luego pasar los detectores de pose solamente sobre los robots detectados. Finalmente, se estima la pose con aquel detector que tuviese mayor confianza. De los 912 Aibo detectados, 657 fueron clasificados en alguna pose ("Frontal", "Trasera" o "Lateral"), de las cuales 519 poses fueron correctamente estimadas (esto implica un 79% de tasa de clasificación). La tabla 4 muestra la matriz de confusión de la estimación de la pose. Los clasificadores "Frontal" y "Lateral" obtienen los mejores resultados, logrando un 90% y 80% de correcta clasificación, respectivamente.

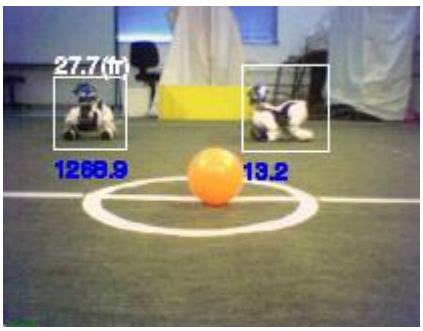
Clase del Robot \ Clase Estimada	AIBO Frontal	AIBO Lateral	AIBO Trasero
AIBO Frontal	91.63 %	11.64 %	33.87 %
AIBO Lateral	3.72 %	81.45 %	15.32 %
AIBO Trasero	4.65 %	6.92 %	50.81 %

Tabla 4: Matriz de Confusión: Estimación de pose de los Aibo detectados

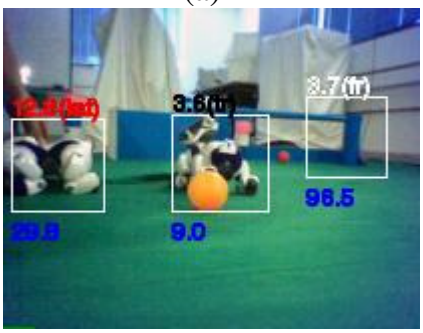
En las siguientes imágenes se pueden ver algunos ejemplos de clasificaciones de pose. Los encuadrados blancos representan detecciones de un robot (con el clasificador frontal), el número en azul debajo de los encuadrados representa la confianza de la detección original y los números arriba del encuadrado muestran la confianza de la pose detectada (cuando existe alguna detección de pose). Además, estos últimos tienen un color que representa la pose detectada: el blanco indica pose frontal, el rojo una pose lateral y el negro una pose trasera. En las fotos se muestran detecciones y clasificaciones correctas, detecciones mal clasificadas, detecciones sin clasificación y falsas detecciones de robots.



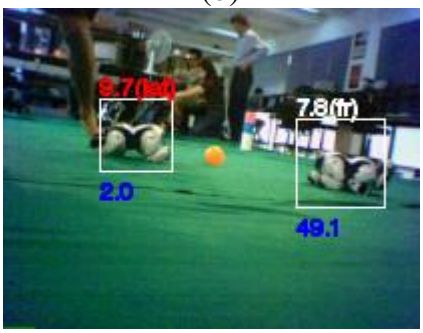
(a)



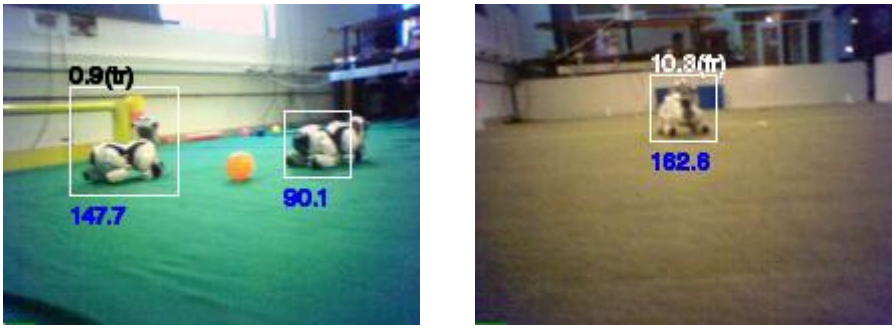
(b)



(c)



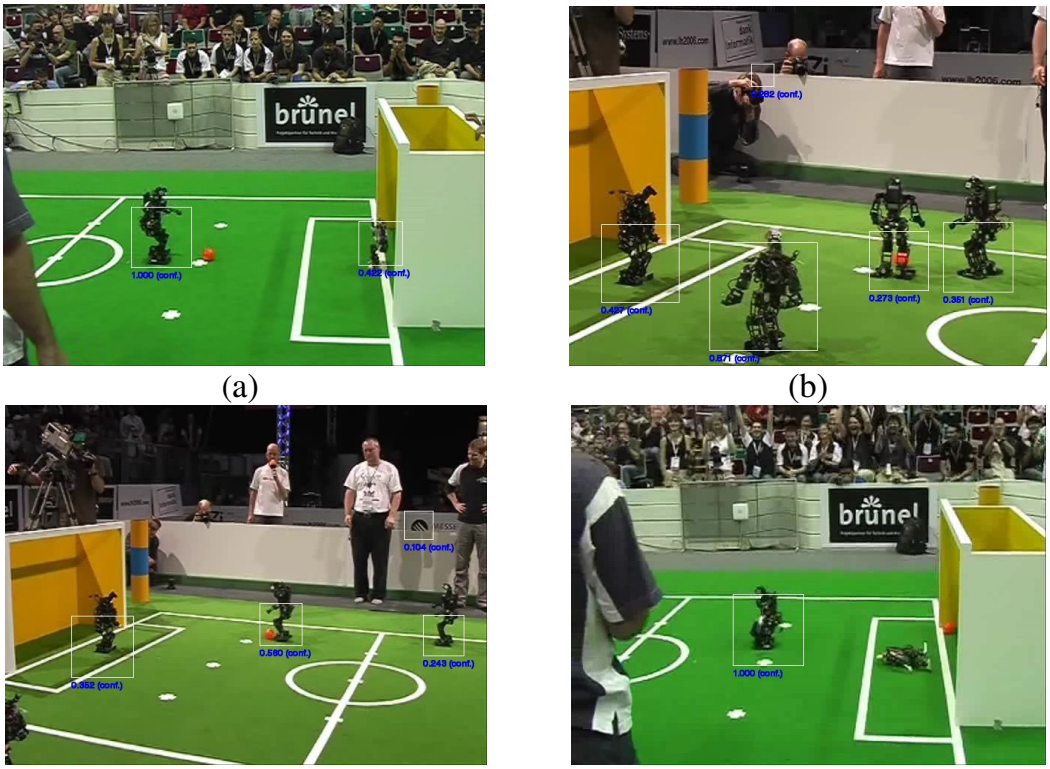
(d)



(e) (f)

Figura 9: Resultados de detecciones de pose sobre la base de datos de Evaluación de los Aibo (AIBODetUChileEval). Se pueden ver clasificaciones laterales (a, c, d), clasificaciones frontales (a, b, f), clasificaciones traseras (c, e), y falsas clasificaciones y detecciones (c, d).

Por otro lado, también se desea probar la eficiencia del detector de Humanoides entrenado. Al igual que el detector de Aibo, éste obtiene resultados bastante satisfactorios ya que se logra una tasa de detección del 92.2% con 123 Falsos Positivos en 244 imágenes. En este caso, también hay que considerar que se usan distintos tipos de Humanoides en el entrenamiento y que la base de datos de prueba es más limitada (en cuanto a condiciones de entorno), ya que no se dispone de tantas imágenes ni videos.



(a) (b)

(c) (d)

Figura 10: Resultados de detecciones sobre la base de datos de Evaluación de los Humanoides (HDetUChileEval).

5.3. **Detección de robots en Aibo**

Luego de obtener todos los resultados se trabaja en implementar el algoritmo en el robot Aibo e integrarlo al código de juego de UChile1, para probar su utilidad práctica en condiciones reales. Los robots Aibo tienen su propio sistema operativo, pero existen herramientas de compilación para correr programas escritos en C y C++ (entre otros). De todas formas, hay que modificar el código implementado anteriormente para adecuarlo a la estructura del sistema operativo de los Aibo (que se basa principalmente en el paralelismo entre varias tareas).

Para programar en los Aibo se ocupa Open-R, que es una arquitectura de programación que corre sobre el sistema operativo Aperios de Sony. Aperios está orientado a mensajes cooperativos multi-tareas, es decir, corre varios procesos simultáneamente, que se comunican a través de mensajes. El código de juego de UChile1 aprovecha este paralelismo para correr varios módulos simultáneamente. Estos módulos son WorldPerceptor, que se encarga de la visión y localización; Engine, que se preocupa de la motricidad y los movimientos del robot; UDPCommunication, que se encarga de los mensajes entre robots; y GameController, que recibe ordenes de juego (comienzo de partido, penalización, etc…). El sistema Open-R permite crear o eliminar procesos, determinar que mensajes se envían y reciben, y también controlar la prioridad de cada proceso que se está corriendo en relación a los demás, lo que permite darle mayor o menor procesamiento a ciertas tareas. Esto permite controlar la importancia que se le va a dar a la detección de robots mediante Adaboost en el código y de esta forma determinar la velocidad a la que corren las detecciones.

En un principio, se programa la plataforma de detección desarrollada directamente en el modulo WorldPercetor, ya que ese es el que trabaja con la imagen que ve el robot. Sin embargo, se aprecia una notoria demora en procesar la imagen en el Aibo con todos los parámetros de detección al máximo, además de una notoria disminución en la capacidad

de juego. Luego se modifican algunos parámetros de la detección, para acelerar el proceso. En general la detección de robots en Aibo a través de Adaboost seguía siendo muy lenta para ser considerada en tiempo real, y se crean varias soluciones para arreglar este problema. Las principales fueron ejecutar la detección sólo en momentos claves, efectuarla solo cada cierto tiempo, o reducir el área de búsqueda en la imagen. Esto permite seguir teniendo la ventaja de tener las detecciones de robots sin perder la reactividad de juego.

Por otra parte, se decide crear un módulo nuevo dedicado únicamente a la detección de robots. Esto mejora notablemente la velocidad de reacción del robot y permite controlar un frame-rate de detección independiente al efectuado por el proceso de WorldPerceptor. Además, con el control sobre la prioridad de los procesos se puede sintonizar un equilibrio entre estos procesos, permitiendo tener detección de robots cada cierto tiempo, manteniendo un '*frame-rate*' elevado en WorldPerceptor. Esto, junto con la reducción del área de búsqueda (por ejemplo no procesar lo que esta más arriba del horizonte visual), son las principales mejoras que permitieron ocupar la detección de robots durante el juego. El módulo de detección de robots envía un mensaje a WorldPerceptor cada vez que termina su proceso con la posición del robot en la imagen (si es que hay uno), y cuando recibe la confirmación de que llegó el mensaje parte el procesamiento de la imagen que se encuentra actualmente en la cámara.

Aún así hay varios factores que mejorar. Al tener la información de un robot, cada cierto tiempo se debe hacer un uso inteligente de ésta para saber donde está el robot detectado, ver si es posible determinar su velocidad, y dónde va a estar en los próximos instantes en los cuales no se va a recibir detecciones. También hay que seguir mejorando la velocidad de procesamiento, de manera a obtener la mayor cantidad de detecciones por segundo y, de esta forma, poder hacer cosas más complejas con un mapa global de los robots en la cancha.

La posibilidad de detectar robots durante un partido de fútbol robótico permite elaborar estrategias de juego más avanzadas. Se puede por ejemplo esquivar rivales, tirar al arco evitando al arquero, elaborar estrategias de pases, entre otras cosas. Con esto se pueden programar mejores estrategias de posicionamiento, de coordinación entre jugadores y de ataque, lo cual lleva a tener un nivel de juego más avanzado y, por ende, permite tener un mejor desempeño en general.

5.4. **Tiempos de Procesamiento**

Para analizar los tiempos de procesamiento se realizan pruebas en un computador portátil y dentro del sistema operativo del Aibo. Los Aibo son modelo ERS7 y contienen un procesador RISC de 64 bits (MIPS R7000) a 576 MHz, con 64MB de RAM. La cámara que ocupan es a color y entrega imágenes de 208*160 píxeles a 30 imágenes por segundo. El computador portátil usado tiene un procesador Intel Core Duo de 1.73 MHz y 1GB de RAM, corriendo Windows XP. Las pruebas en el Aibo se hicieron corriendo el código de juego de UChile1 que se ocupa para fútbol robótico, mientras que en el computador portátil el procesador estaba dedicado casi exclusivamente a detectar robots (las imágenes procesadas también eran de 208*160 píxeles).

La velocidad de procesamiento depende principalmente del factor de escalamiento, y del número de escalas que se salta el algoritmo antes de comenzar el procesamiento de la imagen (factores explicados anteriormente donde se describe el sistema utilizado). La tabla 5 muestra los resultados obtenidos. El detector sigue funcionando correctamente con un factor de escalamiento de 1.2 y saltándose hasta los 2 primeros escalamientos, con lo cual obtiene 2.2 imágenes por segundo en el Aibo y 12.5 en el computador portátil. El tiempo de procesamiento es bastante elevado en los Aibo y es necesario usar distintas estrategias de optimización para poder usar el detector en situaciones reales de juego. Para esto, se puede hacer el procesamiento cada cierto tiempo (90 a 200 milisegundos) o saltarse cierta cantidad de imágenes, por ejemplo.

Configuración	Imágenes por segundo en computador portátil	Imágenes por segundo en robot Aibo
Escalamiento de 1.15 - sin saltarse escalas	3,4	0,7
Escalamiento de 1.15 - saltándose la 1era escala	6,7	1,1
Escalamiento de 1.15 - saltándose las escalas 1 y 2	9,1	1,3
Escalamiento de 1.15 - saltándose las escalas 1,2 y 3	11,1	1,9
Escalamiento de 1.2 - sin saltarse escalas	4,8	0,8
Escalamiento de 1.2 - saltándose la 1era escala	9,1	1,6
Escalamiento de 1.2 - saltándose las escalas 1 y 2	12,5	2,2
Escalamiento de 1.2 - saltándose las escalas 1,2 y 3	16,7	2,9

Tabla 5. Tiempo de procesamiento del detector Frontal de robots Aibo.

5.5. **Árbitro Computacional**

Aparte de realizar la evaluación práctica de los detectores, se decide implementar un árbitro computacional que ocupase la plataforma generada para detectar robots móviles.

El árbitro robótico consiste en usar un robot de servicio para arbitrar un partido de fútbol robótico. El robot tiene una cámara para analizar el partido y un sistema de locomoción para ir siguiendo las acciones. Las imágenes son procesadas en un computador a través de un sistema de análisis de juego que procesa hasta 10 imágenes por segundo. Este sistema de análisis contiene detectores de pelota, landmarks y líneas, además del detector de robots. Con esta información, el robot toma decisiones de arbitraje con respecto a lo que está ocurriendo en la cancha.

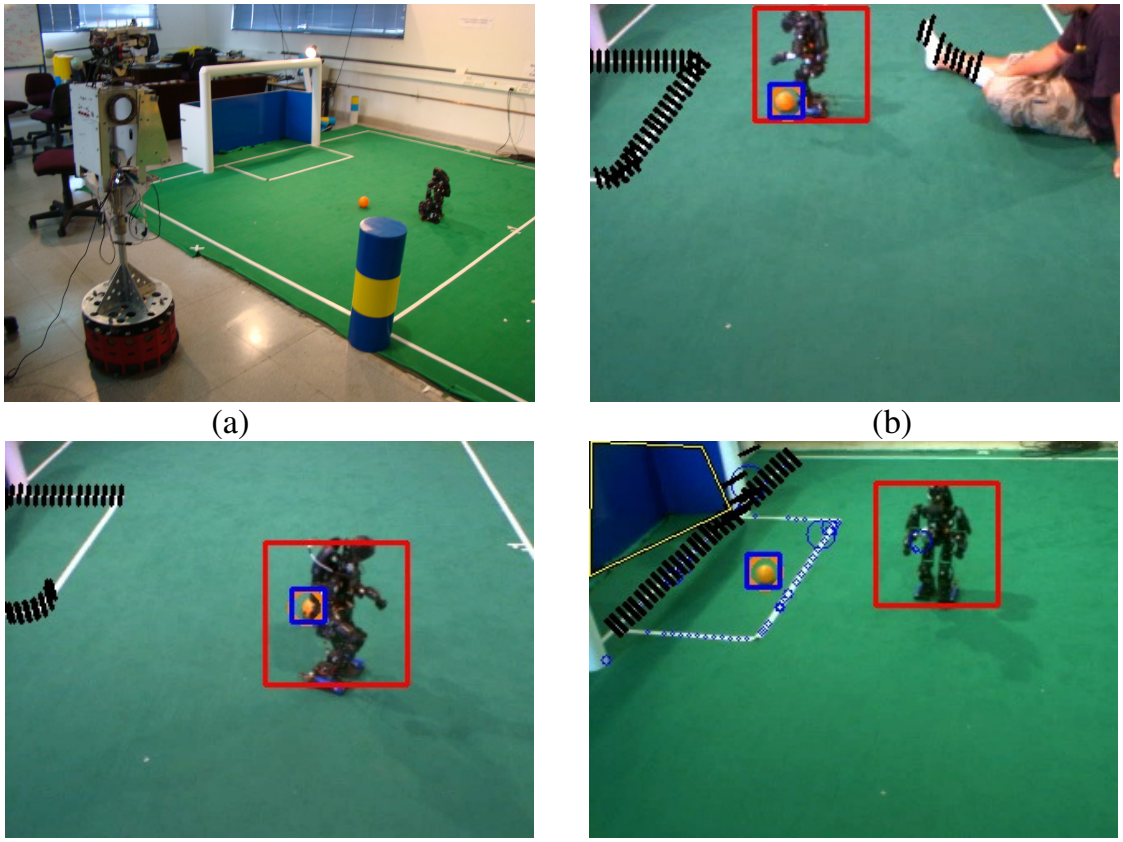
Esta aplicación muestra lo útil que puede ser la detección de robots, ya que gracias a esto el árbitro computacional puede tomar muchas decisiones de juego de manera

independiente. Actualmente se puede detectar cuando un jugador está dentro o fuera de la cancha, cuando tiene la posesión de la pelota, cuál fue el último en tener la pelota, además de la posición de el (o los) jugador(es) en la cancha, entre otras cosas.

Para esta aplicación se ocupa, además, un sistema de seguimiento para los robots basado en el algoritmo de Mean Shift [20]. Este algoritmo permite seguir la posición de objetos no rígidos en imágenes, pero para esto debe tener la posición inicial del objeto a seguir. Esto permite acelerar el procesamiento del sistema de análisis de juego, ya que el sistema de seguimiento a través de Mean Shift es notoriamente más rápido de ejecutar que la detección de robots con Adaboost. En la implementación final, el detector de robots se ocupa una vez cada diez imágenes y en el resto se usa el sistema de seguimiento (sin embargo, esto es muy fácil de modificar).

Las detecciones de robots obtenidas en el árbitro computacional son muy alentadoras. Por ahora hay muy pocos resultados cuantitativos en cuanto a la detección de robots, ya que el trabajo principal del árbitro abarcaba muchos otros aspectos. Sin embargo, en las imágenes de prueba se logran tasas de detecciones mayores al 95% con menos de un falso positivo cada 10 imágenes. Hay que considerar que estas pruebas se obtienen en condiciones de laboratorio, y que no hubo gran variedad de entornos distintos (color de la cancha, fondo, iluminación, etc…).

La alta tasa de detección de esta aplicación se debe a la combinación del detector y del sistema de seguimiento con Mean Shift. Este último tiene un excelente desempeño para seguir objetos no-rígidos, y es muy difícil que pierda al robot una vez que el Adaboost le entrega una detección. Con esto además se comprueba que el sistema de seguimiento ("*tracking*") funciona correctamente y permite un desempeño cercano al tiempo real incluso corriendo distintos algoritmos en paralelo.



(a) (b)

(c) **(d)**

Figura 11: Imágenes del Arbitro Computacional. (a) muestra una vista exterior. (b), (c) y (d) muestran distintas imágenes de juego procesadas.

6. **Conclusiones y Comentarios**

En esta tesis se presenta el estudio e implementación de un detector de robots móviles en imágenes. Se realiza la construcción de distintos detectores para los robots Aibo, además de otro detector para robots de tipo Humanoide. Para esto, se crean varias bases de datos de imágenes de los robots que luego se usan para entrenar los detectores mediante el algoritmo de Adaboost. Luego, estos detectores son probados en un par de aplicaciones distintas para comprobar su funcionamiento y utilidad. Además, al ser enteramente programado en C y C++ permite ser implementado en casi cualquier tipo de plataforma (como es el caso de los Aibo).

El primer paso fue crear una base de datos de entrenamiento la cual necesita de mucho cuidado en incluir la mayor cantidad posible de variaciones en luminosidad, contrastes, y fondos. Esto permite un mejor desempeño del algoritmo ya que sumado a la versatilidad de los clasificadores de tipo Adaboost, permite obtener buenos resultados en distintos entornos y condiciones de iluminación. Junto a la base de datos de entrenamiento se creó una base de datos de pruebas (*benchmark*) la cual permite comparar otros tipos de algoritmos y futuras investigaciones en el tema.

El sistema implementado presenta un buen desempeño en general, sobre todo en cuanto a las tasas de detección obtenidas. En la mayoría de los casos se obtienen resultados satisfactorios que pueden ser usados en distintas aplicaciones. Sin embargo, aún existen aspectos que necesitan mejorarse.

La principal ventaja del sistema es la capacidad de mantener el rendimiento pese a los cambios de iluminación, entorno y tamaños de los robots. Además, para el caso de los Humanoides, se logra obtener un detector que funciona incluso para distintos modelos de robots. Por otra parte, tanto la aplicación en el código de juego en los Aibo como el

árbitro robótico muestran ser funciones útiles del sistema implementado. En ambos casos se obtienen buenos resultados prácticos, lo que demuestra su versatilidad.

En cuanto a las falencias que presenta, la más problemática es el tiempo de procesamiento del algoritmo. Pese a que se logra ocupar en tiempo real, el desempeño del detector baja significativamente y, además, presenta mucha carga de procesamiento lo cual reduce también el desempeño de otros procesos que se estén ejecutando paralelamente. Por ende la intención inicial de ocupar la detección en tiempo real usando sólo Adaboost no resulta y se necesita el uso de otras estrategias para lograrlo.

Una de las tareas más importantes a futuro sería reducir el tiempo de procesamiento del algoritmo de detección. Para esto existen varias opciones: se puede optimizar el código que realiza las detecciones a partir de las cascadas; se puede reducir el área de búsqueda del algoritmo en la imagen; y se pueden optimizar los parámetros de búsqueda del algoritmo según cada aplicación, entre otras cosas.

Sin embargo el uso de un algoritmo de seguimiento como lo es el de *mean-shift* permite un funcionamiento mucho más rápido para conocer la posición de los robots en la imagen. Esto hace que se pueda monitorear la posición de los robots en condiciones cercanas al tiempo real incluso teniendo otros algoritmos corriendo en paralelo (como lo es en el caso del arbitro computacional) y manteniendo una buena tasa de detección. La única desventaja del *mean-shift* es que no permite la detección de robots directamente, sino que necesita otro algoritmo que entregue las detecciones iniciales, pero esto se complementa muy bien con la información entregada a través de Adaboost.

Cabe resaltar que los resultados obtenidos fueron publicados en 2 publicaciones distintas en los simposios realizados durante las competencias Robocup de 2007 y 2008. La primera publicación llamada "*Detection of AIBO and Humanoid Robots using Cascades of Boosted Classifier*" [24] expone los resultados obtenidos para las detecciones de Aibos

y Humanoides. La segunda publicación llamada "*A robot referee for robot soccer*" [25] muestra el trabajo realizado para el Árbitro robótico y sus resultados.

La detección de robots es un aporte importante en muchas aplicaciones. Sin embargo es un primer paso a realizar. La utilidad de las detecciones se hace aún más importante cuando se ocupa de forma más elaborada la información que entregan los perceptores. Actualmente está en proceso de elaboración, por ejemplo, tener un mapa de posiciones y velocidades de los robots (sin tener que verlos en cada instante), y diseñar estrategias más complejas (de juego o posicionamiento, etc…). Otra tarea a futuro podría ser generar un detector múltiple que pueda detectar tanto Aibos como Humanoides, e incluso otros robots (generando otros detectores). En este trabajo se ocupa la información de los robots para ciertas tareas, pero sólo de manera simple y directa. Más adelante se podría ocupar esta información de muchas otras maneras y de forma más compleja como, por ejemplo, elaborar estrategias de pases, o el uso de tácticas adaptivas de juego.

En otros ámbitos, la posibilidad de detectar robots también puede ser útil. En aplicaciones de interacción entre humanos y robots, la detección de estos últimos es vital para establecer comunicaciones eficientes. Otra aplicación posible es la interacción entre robots; cualquier tarea que requiera el trabajo de varios robots en conjunto necesita que éstos puedan reconocerse e interactuar entre ellos, para lo cual la detección visual de robots puede ser útil.

7. **Bibliografía - Anexos**

1. Loncomilla, P., Ruiz-del-Solar, J. (2006). "Gaze Direction Determination of Opponents and Teammates in Robot Soccer". *Lecture Notes in Computer Science 4020 (RoboCup 2005),* Springer, 230 – 242.

2. T. Röfer *et al.*, *German Team 2005 Technical Report*, RoboCup 2005, Four-legged league. Disponible desde Febrero del 2006 en: <http://www.germanteam.org/GT2005.pdf >.

3. M. J. Quinlan *et al.*, *The 2006 NUbots Team Report*, RoboCup 2006, Four-legged league. Disponible desde Febrero del 2007: <http://www.robots.newcastle.edu.au/publications/NUbotFinalReport2006.pdf >.

4. M. Veloso *et al.*, CMDash 2005 Report, Robocup 2005, Tour Legged League. Disponible desde Febrero del 2006 en: <http://www.cs.cmu.edu/~robosoccer/legged/reports/CMDash05-report.pdf >.

5. J. Ruiz-del-Solar, P. Loncomilla, and P. Vallejos, "An automated refereeing and analysis tool for the Four-Legged League". *Lecture Notes in Computer Science* 4434 (*RoboCup 2006*), Springer, pp. 206-218, 2007.

6. R. Verschae, J. Ruiz-del-Solar, M. Correa, "A Unified Learning Framework for object Detection and Classification using Nested Cascades of Boosted Classifiers", *Machine Vision and Applications*, vol 19, No 2, pp 85-103, Enero 2008.

7. R.E. Schapire, Y. Singer, "Improved Boosting Algorithms using Confidence-rated Predictions", *Machine Learning*, 37(3):297-336, 1999.

8. P. Viola, M. Jones, "Rapid object detection using a boosted cascade of simple features", *Proc. IEEE Conf. on Computer Vision and Pattern Recognition*, pp. 511 – 518, 2001.

9. Y. Freund, R. Schapire, "A Short Introduction to Boosting". Journal of Japanese Society for Artificial Intelligence, 14(5):771-780, en Septiembre 1999.

10. P. Viola and M. Jones, "Fast and robust classification using asymmetric adaboost and a detector cascade", *Advances in Neural Information Processing System 14*. MIT Press, 2002.

11. M. J. Jones and P. Viola, "Face Recognition Using Boosted Local Features", Mitsubishi Electric Research Laboratories*, technical report TR2003-25*, April 2003, disponible en Agosto del 2005 en <http://www.merl.com/publications/TR2003-025/ >.

12. P. Viola and M. Jones, "Robust Real Time Object Detection", Second International Workshop on Statistical And Computational Theories of Vision – Modeling, Learning, Computing and Sampling, Julio 2001.

13. Sun J., Rehg J.M, Bobick A.F, (2004) "Automatic Cascade Training with Perturbation Bias", *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2004)* vol. 2, pp. 276-283.

14. Xiao R., Zhu L., Zhang H.-J. (2003), "Boosting chain learning for object detection", *9th IEEE Int. Conf. on Computer Vision*, Vol. 1, pp. 709- 715.

15. T. Laue, T. Röfer, "Integrating Simple Unreliable Perceptions for Accurate Robot Modeling in the Four-Legged League", *Lecture Notes in Computer Science 4434 (RoboCup 2006),* Springer, 474-482.

16. Yang M., Kriegman D., Ahuja N. (2002), "Detecting Faces in Images: A Survey", *IEEE Trans. Pattern Analysis and Machine Intelligence*, Vol. 24, No 1, pp. 34-58.

17. Freund, Y. and Schapire, R.E. 1995. A decision-theoretic generalization of on-line learning and an application to boosting. In *Computational Learning Theory: Eurocolt 95*, Springer-Verlag, pp. 23–37.

18. <http://www.robocup.cl/VideosHumanoides/>, Bases de dato de prueba para comparación de resultados (Revisado por última vez en Septiembre 2008).

19. Sung K., Poggio T. (1998), "Example-Based Learning for Viewed-Based Human Face Detection", *IEEE Trans. Pattern Anal. Mach. Intell.*, Vol.20, No. 1, 39-51.

20. N.S. Peng, J. Yang, and Z. Liu, Mean shift blob tracking with kernel histogram filtering and hypothesis testing, *Pattern Recognition Letters*, vol. 26, pp. 605-614, 2005.

21. Verschae R., "Detección de caras en bases de datos de imágenes", Tesis (ingeniero civil electricista), Facultad de Ciencias Físicas y Matemáticas, Universidad de Chile, 2003.

22. <http://www.owlnet.rice.edu/~elec301/Projects99/imcomp/Haar.htm>, información sobre *wavelets* de Haar (Revisado por última vez en Septiembre 2008).

23. <www.robocup.org/>, página oficial de competencia de futbol robótico (Revisado por última vez en Marzo 2009).

24. M. Arenas, J. Ruiz-del-Solar, and R. Verschae, “Detection of Aibo and Humanoid Robots using Cascades of Boosted Classifiers”. *Lecture Notes in Computer Science,* vol. 5001/2007, pp 449-456 (Robocup 2007).

25. M. Arenas, J. Ruiz-del-Solar, S. Norambuena, S. Cubillos, “A Robot Referee for Robot Soccer”. *Lecture Notes in Computer Science*, aceptado (Robocup 2008).