UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE INGENIERÍA ELÉCTRICA

DESARROLLO DE SISTEMAS DE VISIÓN EN FÚTBOL ROBÓTICO

MEMORIA PARA OPTAR AL TITULO DE INGENIERO CIVIL ELECTRICISTA

RICARDO DODDS ROJAS

PROFESOR GUÍA:
JAVIER RUIZ DEL SOLAR SAN MARTÍN

MIEMBROS DE LA COMISIÓN:
PABLO GUERRERO PÉREZ
HECTOR AGUSTO ALEGRÍA

SANTIAGO DE CHILE
SEPTIEMBRE 2009

AGRADECIMIENTOS

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

TABLA DE CONTENIDOS

TABLA DE ILUSTRACIONES

RESUMEN DE LA MEMORIA
PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL ELECTRICISTA
POR: RICARDO DODDS ROJAS
FECHA: 05/09/09
PROF. GUIA: DR. JAVIER RUIZ DEL SOLAR

"DESARROLLO DE SISTEMAS DE VISIÓN EN FÚTBOL ROBÓTICO"

Esta memoria tiene como objetivo diseñar e implementar algoritmos de visión computacional, en diferentes etapas del sistema de visión de un robot que juega fútbol robótico. Dado que por lo general un robot tiene una capacidad de procesamiento limitada, el principal objetivo de esta memoria es mejorar la eficiencia del sistema completo, sin afectar su rendimiento. Las mejoras propuestas consisten básicamente en tres etapas: adaptación del sistema de visión al uso de un lente gran angular, implementación de un perceptor visual del arco de fútbol basado en un clasificador construido a partir de una red neuronal y el diseño de un sistema de procesamiento de imágenes multi-resolución.

Para adaptar el sistema de visión al uso de un lente gran angular se utiliza un mapeo desde el espacio de la imagen con gran angular a un espacio que asume proyección plana en la imagen, basado en una transformación polinomial. Los parámetros de este mapeo corresponden a los coeficientes del polinomio de transformación y son determinados a partir de un algoritmo de optimización *PSO (Particle Swarm Optimization)*, que utiliza como función objetivo una medida de distorsión de la imagen mapeada.

Para el perceptor del arco se utilizan en una primera etapa una serie de reglas binarias para descartar rápidamente percepciones erróneas. Posteriormente se extraen características de los candidatos que se utilizan como entrada a un clasificador basado en una red neuronal del tipo *MLP (Multi Layer Perceptron)*. Así los candidatos reciben un puntaje de acuerdo a sus características y se escoge el mejor.

En el sistema de procesamiento multi-resolución se toma en cuenta el hecho que los objetos más lejanos, y que por lo tanto se ven más pequeños en la imagen, se encuentran cerca del horizonte. A partir de esta premisa se realiza un procesamiento más fino de la imagen en sectores cercanos al horizonte.

Dada la naturaleza del trabajo se utilizan distintas herramientas para medir el desempeño de cada parte. Para el caso del mapeo de la imagen se utiliza el error máximo de alineamiento en la imagen resultante de puntos co-lineales en el mundo real, medido a través de la norma de los residuos arrojados por un ajuste de mínimos cuadrados, obteniéndose un error de ajuste máximo de 1,02. Mientras que para medir el rendimiento del perceptor basado en un clasificador se miden: tasa de detecciones correctas y tasa de falsos positivos, encontrándose un 96,5 y 1,5 [%] respectivamente. Finalmente el resultado del algoritmo multi-resolución se evalúa a través del cálculo del máximo número de pixeles recorridos al procesar una imagen, con lo cual se determina que se analizan en el peor de los casos, menos de un tercio de los pixeles que usando el sistema anterior.

Los resultados obtenidos muestran que los métodos propuestos tienen un buen desempeño. En cada etapa se aprecia la ventaja de su uso, por lo que se considera el trabajo realizado como una mejora integral al sistema de visión del equipo de fútbol robótico. No obstante, no deja de ser interesante buscar mejores aproximaciones a cada problema y al sistema completo.

I. INTRODUCCIÓN

Este trabajo se enmarca en el contexto del fútbol robótico, y se desarrolla dentro del equipo de robótica *UChile*, del Departamento de Ingeniería Eléctrica de la Universidad de Chile. El trabajo que se expone a continuación se presenta como un intento de mejorar el sistema de visión de los robots bípedos, interviniendo en distintas etapas de este proceso.

1.1 MOTIVACIÓN

Aunque para nosotros sea natural, el medio en que nos desenvolvemos sufre a diario infinitas modificaciones a las cuales debemos adaptarnos. Es necesario para esto ser capaces de percibir el entorno de una manera adecuada. Gran cantidad de la información que recibimos del mundo exterior es captada por la vista. Así, utilizamos nuestros ojos constantemente hasta para resolver tareas muy sencillas: calcular la altura del próximo peldaño de la escalera, esquivar a una persona que se cruza en nuestro camino, agarrar la taza de café sobre el escritorio, etc; tareas todas cotidianas pero muy difíciles de resolver sin retroalimentación perceptual.

De forma similar, para un robot que quisiera desenvolverse de forma autónoma en un ambiente parecido al de los humanos, probablemente sea indispensable contar con un sistema de percepción visual elaborado. Por esta razón el desarrollo de dispositivos electrónicos y algoritmos de visión computacional resulta cada vez más interesante.

En el fútbol robótico se genera una instancia interesante ya que éste simula una actividad compleja realizada por las personas, dentro de un ambiente controlado. Aunque en una primera impresión sea percibido simplemente como un juego, es en realidad un nicho donde se trabaja en importantes avances dentro del ámbito de la robótica.

Es de particular interés el desarrollo de sistemas de visión en tanto permiten a un robot percibir su entorno y poder interactuar con él. Si bien es cierto existen diferentes

instrumentos con los cuales se pueden medir variables de interés del medio, las imágenes aportan información indispensable en medios como el descrito anteriormente.

1.2 UCHILE

El equipo de robótica *UChile* es un proyecto que pertenece al Departamento de Ingeniería Eléctrica de la Universidad de Chile y se avoca al estudio e investigación de la robótica. La constante participación del equipo en competencias internacionales de robótica permite por una parte compartir nuevos conocimientos en este campo, además de comparar el nivel entre universidades según el desempeño en los juegos. Estas participaciones se complementan además con actividades científicas (publicación de papers, proyectos industriales) y desarrollo de programas educacionales para niños.

Desde el año 2003 el equipo *UChile* ha participado en todas las competencias mundiales de *RoboCup*, en la categoría *SPL*[1] - *Standard Platform League* (Liga de plataforma estándar), incorporando posteriormente (2007) al equipo las categorías de humanoides y @Home. En el año 2008 se aborda como desafío principal la participación en la nueva categoría de bípedos de la *SPL*. Es posible mencionar que durante estos años el equipo ha obtenido importantes reconocimientos.

RoboCup es una competencia internacional orientada al desarrollo de temas relacionados con inteligencia artificial (*IA*) y robótica. Utiliza como plataforma de prueba principal el fútbol robótico. Se desarrollan en paralelo distintas categorías que tienen un objetivo en común: "Para el año 2050, diseñar un equipo de fútbol de robots humanoides capaz de ganarle al equipo de humanos campeones del mundo". Este ambicioso objetivo implica un enorme trabajo y desarrollo tecnológico, que se irán logrando paulatinamente y, como es natural, en el camino se encontrarán aplicaciones en materias que vayan más allá del fútbol robótico. Tal es el caso de los robots de

[1] En un principio en esta categoría se utilizan los robots cuadrúpedos *AIBO* de SONY, los cuales se reemplazan en el año 2008 por los robots bípedos *NAO*, fabricados por Aldebaran, que pasan a ser la plataforma oficial. Actualmente participan ambos robots, por separado, en la *SPL*.

servicio, que tímidamente se han ido incorporando en distintos ámbitos como hospitales, museos y hogares.

La arquitectura actual del software del equipo está basada en librerías originalmente desarrolladas para la categoría de robots cuadrúpedos *AIBO*. Dado que el equipo trabaja en distintas categorías de robots (que sin embargo tienen un objetivo similar), el software ha sido diseñado en dos etapas: la etapa independiente de la plataforma, relacionada con los objetivos de alto nivel, y la etapa dependiente de la plataforma, donde se implementa la comunicación entre el hardware y la etapa independiente de la plataforma. La idea detrás de esto es poder usar el mismo software en todos los robots del equipo, introduciendo la menor cantidad de cambios posibles.

La etapa independiente de la plataforma está dividida en cuatro módulos: visión, localización, estrategia y actuación. Estos módulos interactúan dentro de cada robot, compartiendo además información de estrategia y localización entre los robots del equipo. En la Figura 1 se muestra un diagrama jerárquico de los módulos: abajo los procesos de bajo nivel de visión y actuación. Más arriba los procesos de alto nivel de localización y estrategia.

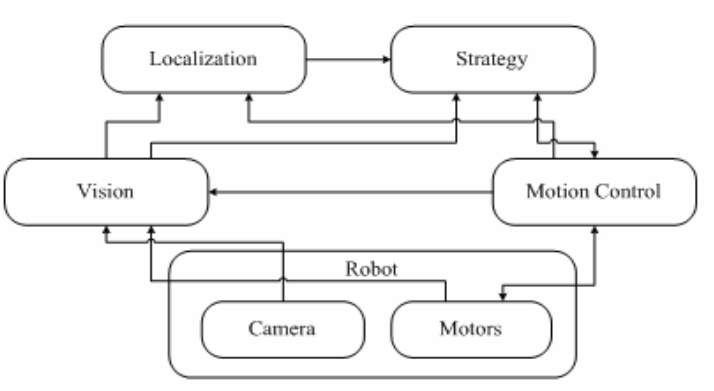

Figura 1. Organización modular del sistema.

Actualmente el equipo *UChile* participa activamente en tres ligas: humanoides, *SPL*, y @Home. En la liga de humanoides (ver Figura 2), participan robots autónomos con cuerpo y sentidos similar a los humanos, que juegan fútbol unos contra otros. Además

de las competiciones de fútbol se realizan desafíos técnicos. Los robots se dividen en dos clases según su tamaño: KidSize (30-60cm de altura) y TeenSize (100-160cm de altura). Caminata dinámica, correr y patear la pelota mientras mantiene el equilibrio, la percepción visual de la pelota, de otros jugadores y el campo, la auto-localización, y el juego en equipo son algunos de los tópicos de investigación involucrados en la liga de humanoides.

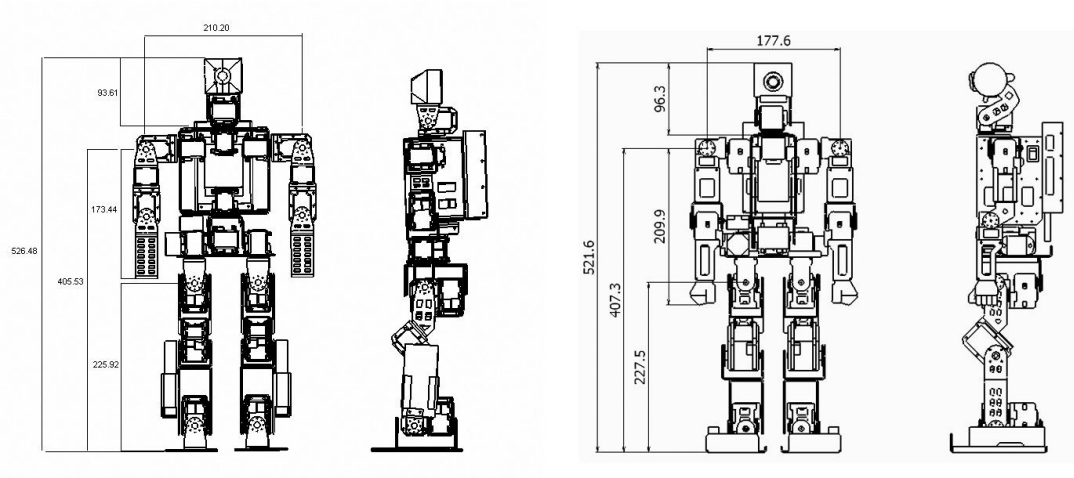

Figura 2. Vista frontal y lateral del robot Hajime HR18 (izquierda) y UCH H1 (derecha) representes del equipo *UChile.*

En la liga SPL todos los equipos utilizan robots idénticos (ver Figura 3). Por lo tanto, los equipos se centran solamente en el desarrollo del software. Los robots funcionan de forma totalmente autónoma, es decir, no hay control externo: ni por los seres humanos, ni por computadores.

Figura 3. Vista frontal robot NAO de Aldebarán.

La categoría Robocup @Home busca desarrollar robots de servicio y asistencia con importantes aplicaciones domésticas. Se utiliza un conjunto de pruebas de referencia para evaluar las capacidades de los robots y su rendimiento en un ambiente hogareño realista. Entre los principales ámbitos de interés se encuentra: interacción y cooperación entre humanos y robots, navegación en un entorno dinámico, visión computacional y reconocimiento de objetos bajo condiciones de luz natural, manipulación de objetos, comportamientos adaptivos e inteligencia ambiental. En la Figura 4 se muestra el robot Bender, desarrollado en el Laboratorio de Robótica de la Universidad de Chile y que compite en la categoría Robocup @Home.

Figura 4. Vista frontal del robot Bender.

Este trabajo será desarrollado para los equipos de humanoides y los bípedos de la *SPL*. La diferencia principal entre estas plataformas son las restricciones del hardware. Como el nombre lo dice, en la *SPL* la configuración de los robots es estándar y no se permite introducir modificaciones, y por lo tanto las diferencias entre equipos se restringen únicamente en la programación del software.

1.3 Objetivo General

Dentro del contexto de fútbol robótico el objetivo general de esta memoria es lograr implementar algoritmos de procesamiento de imágenes que permitan mejorar el rendimiento del sistema de visión de un robot.

1.4 Objetivos Específicos

- Adaptar el sistema de visión para el uso de un lente gran angular.
- Dentro del sistema de percepción del arco, implementar un clasificador basado en una red neuronal feed-forward.
- Diseñar e implementar un algoritmo de visión en multi-resolución.

1.5 Estructura de la Memoria

La memoria se encuentra dividida en 5 capítulos. El primero de estos, es un capítulo introductorio que entrega una visión general del trabajo. El segundo capítulo, entrega los elementos teóricos básicos que permiten una mejor comprensión de la memoria. El tercero, es el cuerpo principal de la memoria, en este se describen el sistema diseñado, su implementación y los elementos que los componen. En el capítulo cuarto, se describen las pruebas realizadas y se analizan sus resultados. Por último, en el quinto capítulo se concluye el trabajo.

II. ANTECEDENTES GENERALES

2.1 TRABAJOS ANTERIORES

2.1.1 Correcciones en Lente Gran Angular

En esta sección se hace una breve revisión de los métodos usados para la calibración de lentes de cámaras. Para corregir la distorsión inherente a un lente gran angular, es importante que cada pixel de la imagen 2D distorsionada sea llevado a su posición original de la escena 3D proyectada en un plano 2D. Debido a la curvatura del lente cada pixel en el plano es cambiado de posición lo que provoca que la imagen se vea distorsionada.

Beck [1] y Miyamoto [2] describen distintos métodos de mapeos que relacionan el ángulo en el plano del mundo real con su correspondiente ángulo en el plano de la imagen. El ángulo en el plano del mundo real es aquel definido entre la línea que conecta un punto en el mundo real con el centro óptico y el eje óptico, mientras que su ángulo correspondiente en el plano de la imagen es aquel definido entre la línea que conecta la proyección de este punto en la imagen con el centro óptico y el eje óptico. Esto se muestra de forma esquemática en la Figura 5. Para el método estereográfico se usa la relación

$$2\,tan\left(\frac{\emptyset'}{2}\right) = \tan(\emptyset)$$

donde $\emptyset'$ en el ángulo en el plano del mundo real y $\emptyset$ el ángulo en el plano de la imagen. En el método de proyección equidistante se usa la relación

$$\emptyset' = \tan(\emptyset)$$

Estos métodos corrigen la imagen cerca del centro óptico, pero hacia los bordes de la imagen la distorsión es más significativa. Anderson, Alvertos y Hall [3] presentan un procediemiento en donde la relación entre $\emptyset'$ y $\emptyset$ se establece mediante un

algoritmo basado en la Ley de Snell. Nuevamente, la transformación es aplicable en un entorno limitado cerca del centro óptico, donde la distorsión es relativamente pequeña.

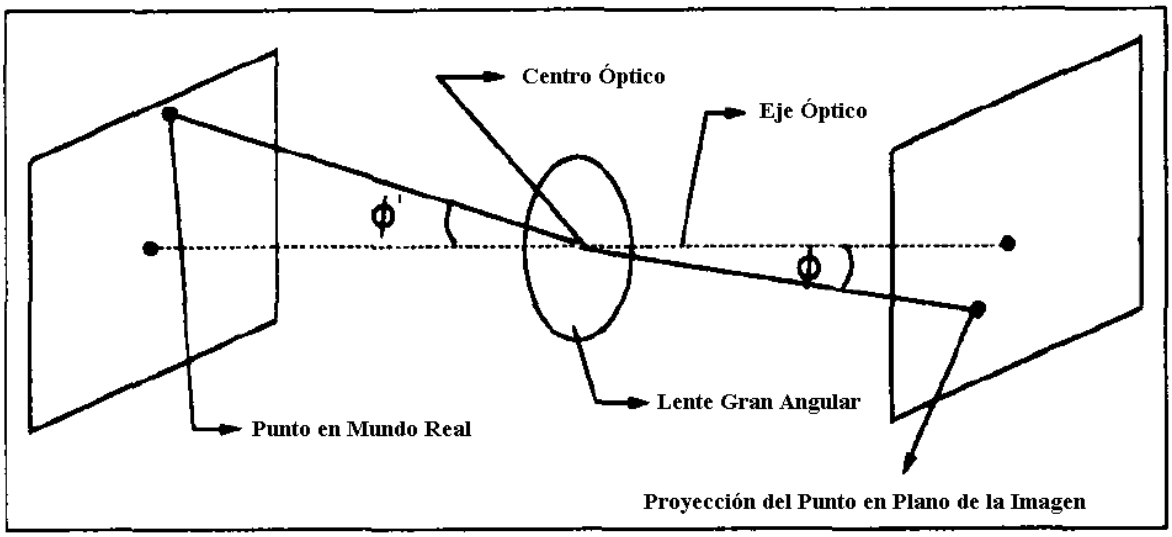

Figura 5: Mapeo del mundo real al plano de la imagen.

Shah y Aggarwal [4] proponen un método basado en transformación polinomial para corregir la distorsión en la imagen. El procediemiento utiliza una transformación polinomial entre el sistema de coordenadas del mundo real y sus posiciones en el plano de la imagen correspondiente. Los coeficientes del polinomio son determinados a través de una estimación Lagrangiana, que utiliza un patrón para medir la distorsión resultante del mapeo.

2.1.2 Percepción en Colores

Debido a que este tema es relativamente extenso y que el objetivo de este capítulo es facilitar la comprensión de los capítulos siguientes, en esta sección se describirá de manera general el módulo de visión que se encuentra implementado actualmente en el equipo *UChile*, y específicamente como está diseñado el perceptor del arco.

A grandes rasgos, el sistema de visión se puede dividir en tres sub-módulos, que son: segmentación de la imagen, formación de regiones o blobs y percepción de objetos de interés.

En la etapa de segmentación de color cada pixel de la imagen es etiquetado en un espacio de colores. Las clases de este espacio están definidas por los colores de los objetos de interés que para este caso son el amarillo, cian, naranjo, blanco, verde, rojo y azul. Los pixeles que no caigan en ninguna de estas clases son clasificados como una octava clase: ningún color. Para realizar la segmentación se utiliza un sistema que usa una *look-up table* (tabla de búsqueda) de 3 dimensiones (ver Figura 6), indexada por las componentes de color del pixel, cada una de las cuales está discretizada en 64 niveles.

Esta tabla es construida en una etapa "off-line", mediante imágenes capturadas desde la cámara del robot, lo más representativas del entorno posibles. Para cada imagen, se marca manualmente las regiones de interés, cada una asociada a una clase de color. Posteriormente se realiza una etapa de generalización para la *look-up table* que consiste aprender una distribución gaussiana para cada clase. Luego en la etapa "on-line", simplemente se consulta la tabla construida.

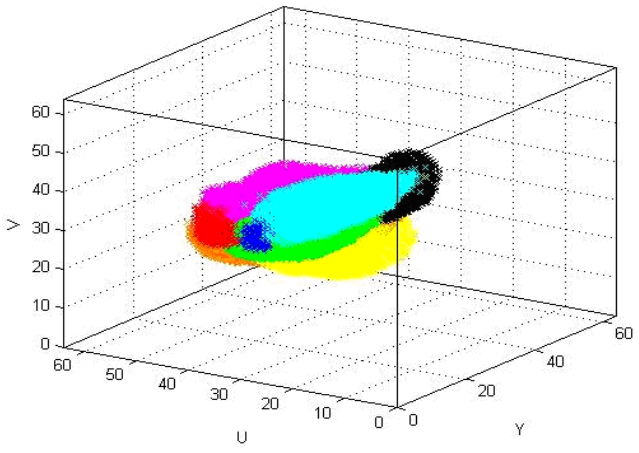

Figura 6. Look-up table para la segmentación.

Para la formación de regiones se genera una codificación *run-length* (tiras) de la imagen representada por un arreglo de 4 x N elementos, donde N es el número de segmentos extraídos de cada imagen. Un segmento consiste en un conjunto de pixeles conectados, del mismo color que pertenecen a la misma fila de la imagen. Esta representación es útil para reducir la cantidad de información y simplificar las siguientes etapas de visión. Posteriormente se inicia un proceso de etiquetado de las regiones conectadas. Para esto se compara el color de los segmentos de cada fila con el de los segmentos adyacentes superiores. Si el color es el mismo se copia la etiqueta correspondiente. Si un segmento tiene dos vecinos superiores del mismo color, pero con distintas etiquetas, éstas son reemplazadas recursivamente y por toda la rama. Si un segmento no tiene etiqueta, se genera una nueva.

Una vez identificadas todas las regiones se caracterizan, determinando para cada una su centro de masa, el tamaño de la región y ocho puntos que corresponden a la intersección entre el borde de la región y las curvas de nivel en las direcciones que se muestran en la Figura 7, en otras palabras son los puntos de más arriba, más abajo, más a la izquierda, más a la derecha y más lejanos en las diagonales de las regiones. Estos descriptores son usados más adelante para el reconocimiento de objetos de interés en el campo de juego.

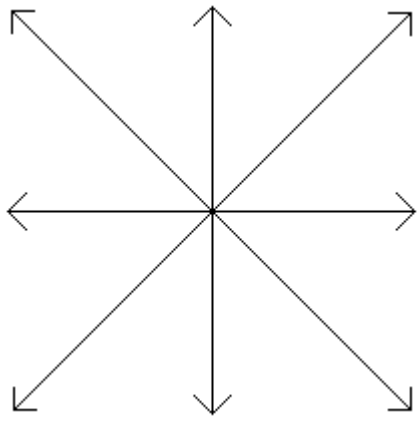

Figura 7. Direcciones de puntos característicos.

La percepción de objetos se realiza mediante un sistema basado en confidencias. Para este trabajo es de particular interés el método de percepción del arco, que será descrito a continuación.

El sistema de percepción se compone de dos etapas. En la primera se descartan rápidamente las regiones candidatas que no cumplen las condiciones mínimas, a través de un filtro de reglas binarias. Con esto se reduce el número de regiones que serán analizadas posteriormente con más detalle, lo que favorece a la eficiencia del sistema de visión. Es importante en esta parte no eliminar candidatos que tengan alguna posibilidad de ser el objeto buscado, por lo que las reglas solo exigen condiciones básicas. Para encontrar las reglas se utiliza un algoritmo genético, que selecciona las reglas y ajusta los parámetros [5].

En la segunda etapa, se analizan las características propias del objeto y se entrega un puntaje ("confidencia") de acuerdo al resultado obtenido. Esto hace posible seleccionar más de un candidato y finalmente elegir el que tenga un mayor puntaje asociado. Las características que determinan los puntajes, se definen de forma empírica, de forma de que su comportamiento sea representativo de la credibilidad de la región candidata.

2.2 ALGORITMO PSO

PSO – Particle Swarm Optimization (Optimización de Enjambre de Partículas) es un algoritmo de optimización, que consiste en un proceso iterativo y estocástico que opera sobre un cúmulo de partículas, donde la posición de cada partícula representa una solución potencial al problema que se está resolviendo. Esta técnica de optimización está inspirada en el comportamiento social de animales tales como una bandada de pájaros o un cardumen de peces. El argumento principal de este algoritmo se puede expresar como: los individuos que son parte de una población tienen una opinión influenciada por la creencia global compartida por todos los individuos que la conforman.

PSO ha sido aplicado con éxito en diferentes campos de investigación para la resolución de problemas de optimización. Algunos ejemplos son: optimización de funciones numéricas [6], entrenamiento de redes neuronales [7] y aprendizaje de sistemas difusos [8].

Para encontrar la solución óptima, el algoritmo actualiza el cúmulo actual de partículas utilizando información acerca de la mejor solución obtenida por cada partícula y la mejor solución obtenida por el cúmulo entero. Cada partícula tiene los siguientes atributos: la velocidad actual, la posición actual, la mejor posición obtenida por la partícula hasta el momento y la mejor posición encontrada por los vecinos de la partícula hasta el momento.

El cúmulo se inicializa generando las posiciones y las velocidades de las partículas de forma aleatoria. Ya inicializado el cúmulo comienza un proceso iterativo, en donde una partícula se mueve desde una posición del espacio de búsqueda hasta otra, simplemente sumando la su velocidad v_i a su posición x_i:

$$x_i \leftarrow x_i + v_i$$

Una vez calculada la nueva posición, se evalúa actualizando el *fitness*[2] (aptitud). Además si el nuevo *fitness* es mejor que el encontrado hasta el momento, se actualizan los valores de mejor posición. El vector velocidad de cada partícula es modificado en cada iteración utilizando la velocidad anterior, la mejor posición conocida por esa partícula y la mejor posición del vecindario. Esto se puede expresar matemáticamente con las siguientes ecuaciones:

$$v_{k+1} = \omega \cdot v_k + \varphi_1 \cdot (pBest_k - x_i) + \varphi_2 \cdot (g_k - x_k)$$

$$x_{k+1} = x_k + v_{k+1}$$

---

[2] El término *fitness* se refiere al valor de aptitud o adecuación de un individuo. Se utiliza para medir que tan buena es una solución para un determinado problema.

donde ω es el factor de inercia, φ_1 y φ_2 son números aleatorios, $pBest_k$ es la mejor posición encontrada por la partícula hasta el momento y g_k es la mejor posición encontrada hasta el momento en el vecindario de dicha partícula.

Esta ecuación modela el movimiento de cada partícula en cada iteración. En la Figura 8 se muestra una representación gráfica del movimiento de una partícula en el espacio de soluciones.

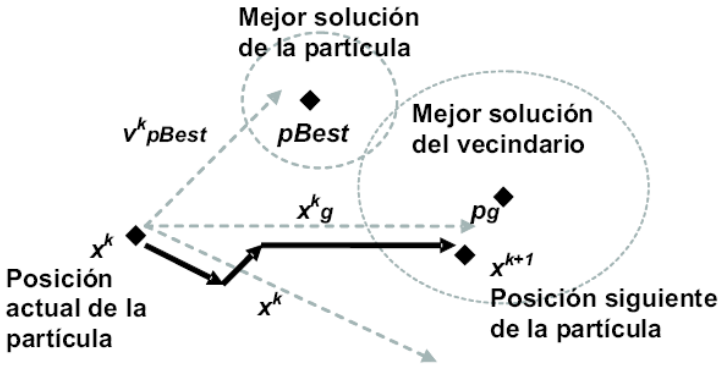

Figura 8. Movimiento de una partícula en el espacio de soluciones.

A continuación se presenta el pseudocódigo de algoritmo PSO básico:

Pseudocódigo del algoritmo PSO básico

S ← *InicializarCumulo*()
while no se alcance la condición de parada[3] **do**
    for i = 1 to *size*(***S***) **do**
        Evaluar cada partícula x_i del cúmulo ***S***
        if *fitness*(x_i) es mejor que *fitness*($pbest_i$) **then**
            $pbest_i \leftarrow x_i$; *fitness*($pbest_i$) ← *fitness*(x_i)
        end if *fitness*($pbest_i$) es mejor que *fitness*($gbest_i$) **then**
            $g_i \leftarrow pbest_i$; *fitness*(g_i) ← *fitness*($pbest_i$)
        end if

[3] La condición de parada se alcanza después de un número dado de iteraciones o al obtenerse un valor establecido para la función de *fitness*.

end for
for *i* = 1 to *size*(***S***) **do**
Elegir $\boldsymbol{p}_j$ la particular con mejor fitness del vecindario $\boldsymbol{p}_i$
Actualizar la velocidad de la partícula $\boldsymbol{p}_i$, de acuerdo con los valores de $\boldsymbol{p}_i$ y $\boldsymbol{p}_j$
Mover la partícula $\boldsymbol{p}_i$ de acuerdo con su nueva velocidad
end for
end while

2.3 REDES NEURONALES FEEDFORWARD

Una red neuronal artificial (*RNA*) es un modelo matemático inspirado en la estructura de las redes neuronales biológicas y en cómo procesan la información. Estos modelos están formados por unidades simples, llamadas neuronas, que se conectan entre sí y se disponen en capas. En la mayoría de las redes neuronales hay una capa de entrada que recibe la información, una capa de salida encargada de transmitir la información procesada al exterior y puede haber una o varias capas ocultas donde se establecen relaciones complejas. Las conexiones entre las neuronas indican la dirección y el sentido en el que fluye la información.

2.3.1 Modelo de Neurona

McCulloch and Pitts en 1943 concibieron un modelo abstracto y simple de una neurona artificial, que es el elemento básico de procesamiento en una *RNA*. En la Figura 9 se muestra su esquema:

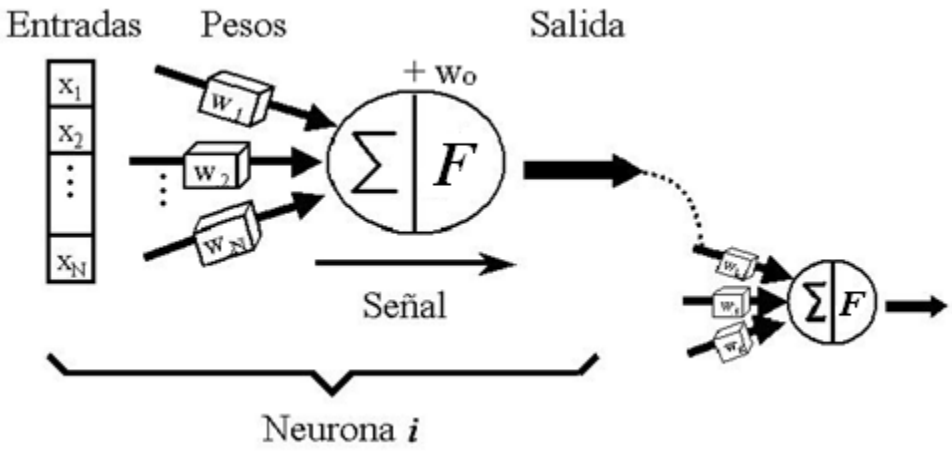

Figura 9. Modelo de una neurona artificial.

El modelo está compuesto por un vector de pesos $\boldsymbol{w} = (\boldsymbol{w_1}, \ldots, \boldsymbol{w_n})^T$ equivalente a las conexiones sinápticas en una neurona real biológica, w_0 ese el umbral de acción o activación, el vector $\boldsymbol{x} = (\boldsymbol{x_1}, \ldots, \boldsymbol{x_n})^T$ es la entrada y el escalar *y* es la salida de la unidad. La actividad consiste en generar una única salida *y* a partir de la aplicación de la función de activación *F* a la suma ponderada entre el vector de entrada *x* y el vector de pesos más un sesgo w_0, obteniéndose la siguiente expresión:

$$y = F\left(\sum_{i=1}^{N} w_i x_i + w_0\right)$$

donde *F* es una función generalmente no-lineal. En un principio la función propuesta por McCulloch-Pitts posee una salida binaria ± 1 conocida como la función de todo o nada que equivale a la función signo dada por:

$$F(x) = sgn(x) = \begin{cases} 1 & x \geq 0 \\ -1 & x < 0 \end{cases}$$

Cuando se consideran neuronas con respuestas de procesamiento gradual, entonces se pueden usar funciones de activación de forma lineal o de forma sigmoidal como la función logística

$$F(x) = \frac{1}{1 + e^{-x}}$$

o la tangente hiperbólica

$$F(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

2.3.2 Estructura de una red neuronal

La arquitectura de una red neuronal es formada conectando múltiples procesadores elementales. Como se mencionó estos elementos están dispuestos en capas que se pueden clasificar en tres: capa de entrada, capas intermedias u ocultas y capa de salida. Basándose en el flujo de información las redes neuronales pueden ser clasificadas en dos: feed-forward, donde la salida de una neurona de una capa, alimenta todas las neuronas de la capa siguiente y redes recurrentes, que poseen conexiones de realimentación, las cuales proporcionan un comportamiento dinámico. Esta estructura forma un sistema adaptivo que posee un algoritmo de aprendizaje para ajustar sus pesos (parámetros libres) para alcanzar los requerimientos de desempeño del problema basado en muestras representativas.

El proceso de aprendizaje más común es supervisado porque para cada *N*-úpla de patrones presentados a la red, el resultado obtenido se compara con el resultado deseado del que también se tiene información temporal. La diferencia entre estos valores se utiliza para ajustar los pesos de la red utilizando un algoritmo iterativo.

Back Propagation es el algoritmo de entrenamiento más común, en el cual iterativamente se retropropaga el error, ajustando el valor de los pesos o conexiones sinápticas. Para esto se realizan en cada iteración dos pasadas: hacia adelante, con los pesos fijos se calcula la respuesta de las unidades ocultas y de salida ante las entradas y posteriormente se determina el error; hacia atrás, la señal de error es propagada hacia atrás usando los pesos de la red y ajustándolos. Un parámetro importante de este algoritmo es la tasa de aprendizaje μ, que determina la velocidad con que se aprende.

Por lo tanto se puede señalar que una *RNA* es un sistema caracterizado por:

- Un conjunto de unidades elementales, cada una de las cuales posee bajas capacidades de procesamiento.

- Una densa estructura interconectada que usa enlaces ponderados

- Parámetros libres que deben ser ajustados para satisfacer los requerimientos de desempeño

Es importante señalar que la propiedad más importante de las redes neuronales artificiales es su capacidad de aprender a partir de un conjunto de patrones de entrenamiento, es decir, es capaz de encontrar un modelo de ajuste de datos.

Los modelos de redes neuronales son dirigidos a partir de los datos, es decir, son capaces de encontrar relaciones (patrones) de forma inductiva por medio de los algoritmos de aprendizaje basado en los datos existentes más que requerir la ayuda de un modelador para especificar la forma funcional de sus interacciones. Es importante por esto, que lo datos sean representativos del universo del problema que se pretende resolver, ya que esta variabilidad determina la capacidad de la red de generalizar las soluciones.

Las redes neuronales ofrecen un método de resolver problemas, de forma individual o combinadas con otros métodos, para aquellas tareas de clasificación, identificación, diagnóstico, optimización o predicción en las que el balance datos-conocimiento se inclina hacia los datos y donde, adicionalmente, puede haber la necesidad de aprendizaje en tiempo de ejecución y de cierta tolerancia a fallos.

III. TRABAJO DESARROLLADO

En este capítulo se presenta una descripción del trabajo desarrollado.

3.1 LENTE GRAN ANGULAR

El uso de un lente gran angular en la cámara del robot, y por lo tanto la introducción de una distorsión en la imagen capturada, requirió la implementación de un sistema de corrección de la imagen. La metodología utilizada para corregir la distorsión está basada en una transformación polinomial, que se aplica on-line.

Los parámetros de esta función de mapeo corresponden a los coeficientes del polinomio, los cuales son difíciles de fijar a mano, por lo que se desarrolló un mecanismo automático para encontrarlos en forma off-line a partir de una foto tomada con el lente gran angular a un patrón de calibración.

Una exacta calibración de la cámara es necesaria en varias aplicaciones que requieran medidas cuantitativas, como visión estéreo, navegación con robots y visión robótica. Por otro lado, el uso de lentes gran angulares es útil cuando se requiere un campo de visión amplio. Aunque estos lentes permiten campos de visión amplios (~180°), introducen en la imagen distorsiones significativas. Esto se puede apreciar al comparar la Figura 10 y la Figura 11.

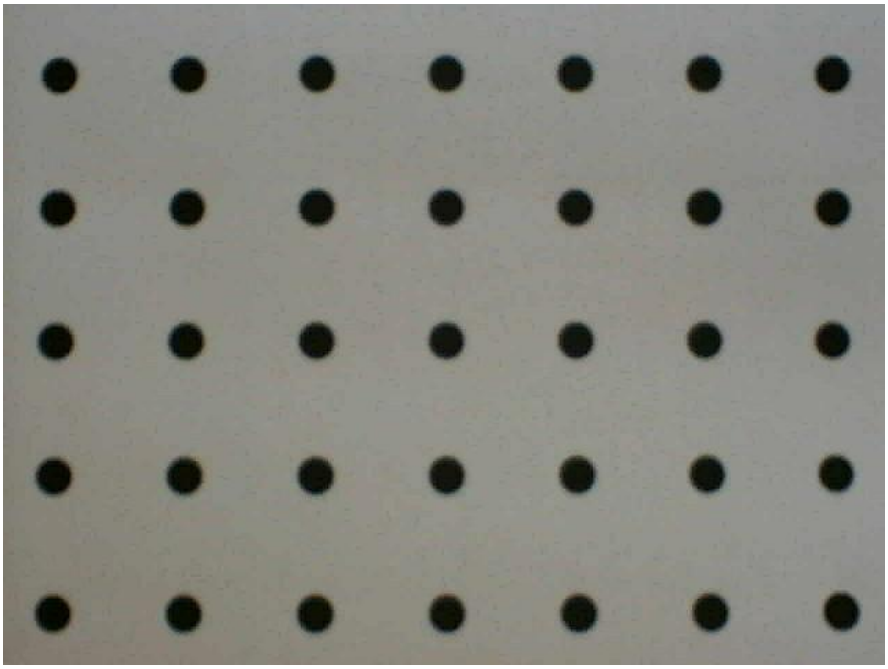

Figura 10: Patrón de calibración del lente gran angular.

La Figura 10 muestra una imagen del patrón de calibración tomada con un lente normal que introduce una distorsión mínima, mientras que en la Figura 11 se muestra el mismo patrón tomado con un lente gran angular. La distorsión resulta en el cambio de la posición de los pixeles desde su posición en la imagen de proyección lineal. Si se desea considerar para cálculos posteriores una geometría de proyección plana es evidente que es necesario corregir la imagen resultante al usar un lente gran angular.

3.1.1 Pre-procesamiento de la Imagen

Una primera parte consiste en el pre-procesamiento de la imagen de modo de poder extraer la información necesaria para las siguientes etapas.

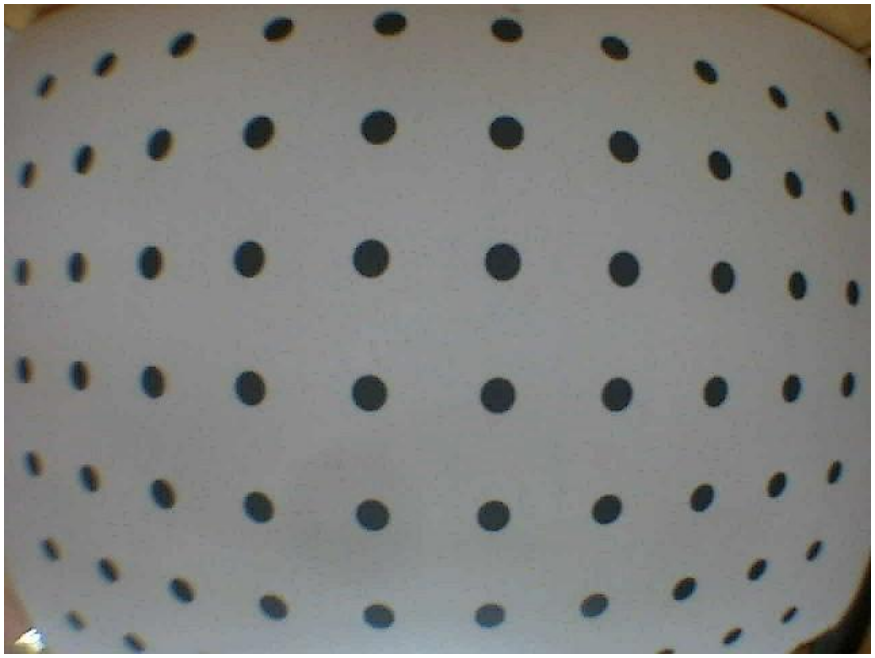

Figura 11: Patrón de calibración capturado a través del lente gran angular.

El procesamiento completo consta de las siguientes etapas:

- Eliminación de información innecesaria.

  Básicamente se limpia la imagen de forma manual eliminando esquinas y puntos de filas o columnas incompletas.

- Binarización.

  En esta parte se transforma la imagen a escala de grises. Luego se determina un umbral de corte para posteriormente binarizar la imagen. Para encontrar el umbral se utiliza el Método Otsu [9], que básicamente consiste en elegir el umbral que minimiza la varianza intra-clase de los pixeles blancos y negros.

- Inversión de colores

  Finalmente, y por motivos prácticos se invierten los colores de la imagen quedando el resultado como se muestra en la Figura 12.

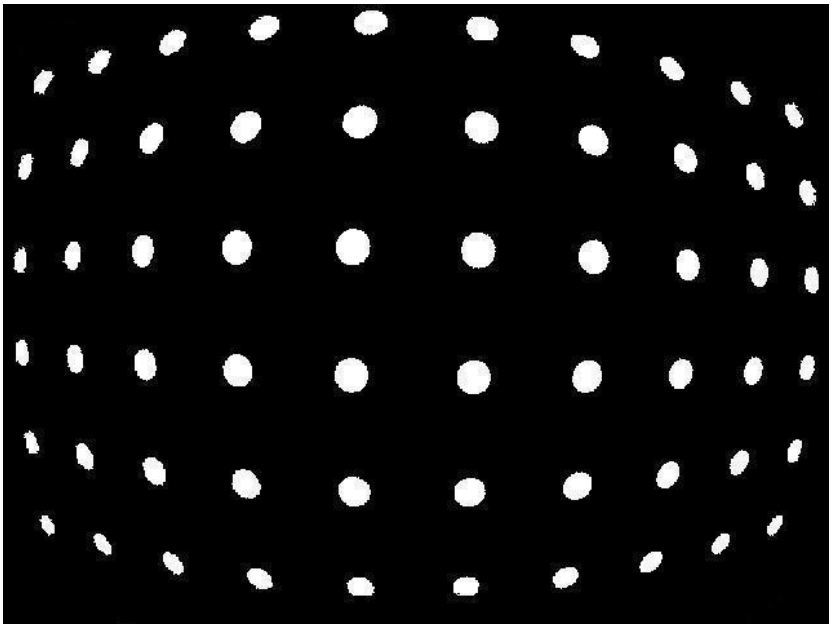

Figura 12: Imagen Pre-procesada.

A partir de la imagen pre-procesada se identifican los puntos como regiones, se calcula la posición de cada uno y finalmente se etiquetan (Figura 13). Para esto se determina el centro de masa de todos los pixeles conexos pertenecientes a la misma región y se aplica un algoritmo de difusión que determina a que fila y columna pertenece cada punto.

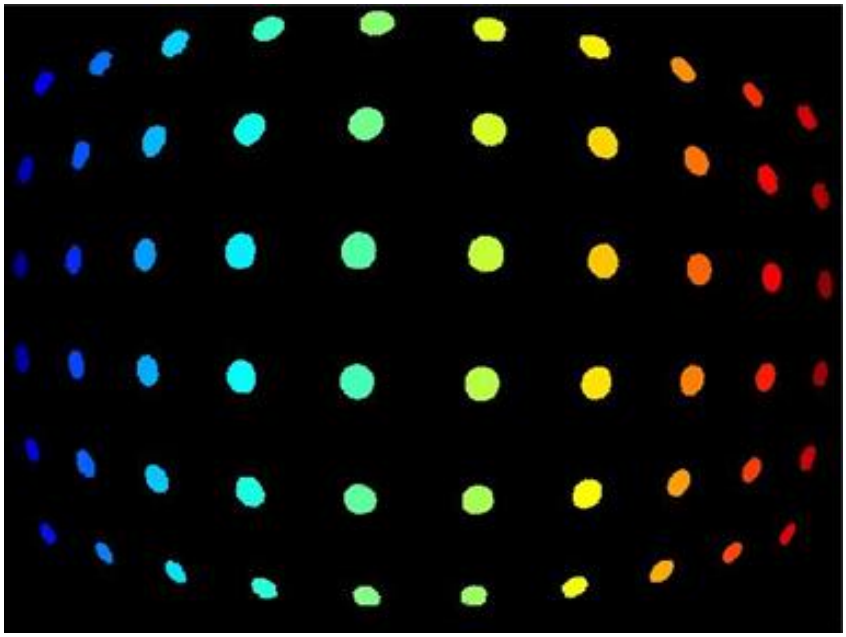

Figura 13: Imagen Etiquetada.

En la Figura 13 se muestran los puntos etiquetados donde cada etiqueta está representada por un color diferente. Esta información se utiliza posteriormente para medir la distorsión de la imagen.

3.1.2 Función de Mapeo

Como se mencionó, el método propuesto está basado en una transformación polinomial. En este caso se utiliza un polinomio de orden cuatro de forma de lograr la precisión necesaria para los cálculos posteriores a partir de la imagen. Polinomios de menor orden no corrigen apropiadamente la distorsión, mientras que polinomios de orden mayor se traducen en un costo de cálculo mayor. Luego, un polinomio de cuarto orden entrega el mejor balance entre precisión y tiempo que toma la corrección. A partir de este polinomio se corrige la posición radial de los puntos medidos a partir del centro óptico. Así, ρ es la distancia medida desde el punto proyectado en el plano de la imagen hasta el centro óptico, mientras que ρ' es la distancia corregida. El modelo de distorsión y las distancias respectivas se muestran en la Figura 14. La ecuación general es

$$\rho' = a \cdot \rho^4 + b \cdot \rho^3 + c \cdot \rho^2 + d \cdot \rho$$

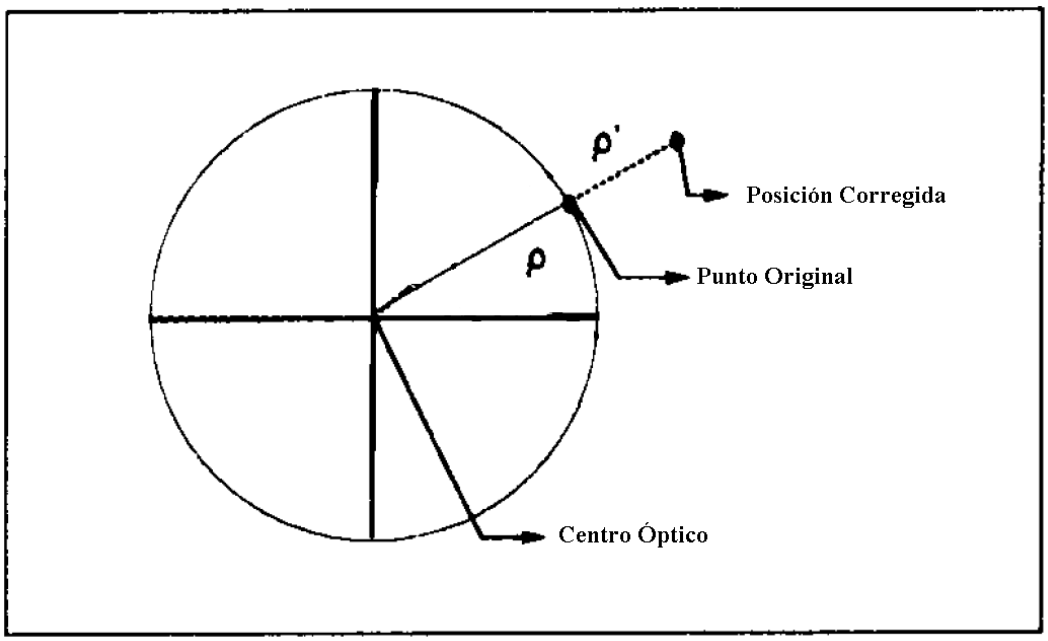

Figura 14: Modelo de distorsión.

donde ρ' es la distancia corregida, ρ es el radio original y a, b, c, d son los coeficientes de distorsión. Como se aprecia en la ecuación anterior no existe término constante ya que la distorsión en el centro óptico es cero.

3.1.3 Optimización de Parámetros por PSO

Este módulo recibe como entrada la información obtenida del procesamiento anterior de la imagen: un arreglo con la posición final de los puntos en la imagen distorsionada, en el cual también se especifican a que fila y columna pertenece cada uno. Para encontrar los coeficientes de distorsión se utiliza el algoritmo de optimización *PSO*, siendo necesario definir una función de *fitness* que permita determinar la calidad de las soluciones.

Como medida de distorsión de la imagen capturada a través del lente gran angular, se usa que los puntos que forman una línea recta en la imagen patrón deberían formar también una línea recta en el plano de la imagen corregida. Como es necesario tener una medida cuantitativa de este hecho, luego de mapear la imagen

con un cierto set de coeficientes de distorsión, se realiza un ajuste de mínimos cuadrados de cada fila y columna de la imagen mapeada. Es posible así definir la función de *fitness* como el máximo error de ajuste obtenido. Para calcular este valor se toma la norma de los residuos de cada uno de los ajustes, y luego se elige el máximo de estos valores.

El algoritmo se ejecuta en *Matlab*, utilizando el toolbox de *PSO*. Para encontrar los valores de los coeficientes de distorsión se define una población de 100 individuos, y se limita a un máximo de 100 iteraciones. Para reducir el espacio de búsqueda se inicializan los parámetros de cada individuo en el intervalo [0; 0,0000001], rango en el cual es sabido se encuentra la solución.

3.1.4 Adaptación del código de visión

Una de las ventajas del método utilizado es que su adaptación al código de visión es relativamente sencilla.

La corrección de la distorsión introducida por el lente gran angular cobra importancia al momento en que se intenta caracterizar un objeto o determinar su posición en la imagen. Luego no es necesario mapear la imagen original completa. La corrección se aplica en una etapa posterior a la segmentación y solo a las regiones interés.

3.2 Perceptor Del Arco

Como se describe en el capítulo anterior, el perceptor del arco está compuesto por dos etapas, la primera de las cuales descarta las regiones que no cumplen las características mínimas a través de un filtro de reglas binarias, mientras que la segunda realiza un análisis de las regiones restantes otorgándole un puntaje a cada una. De esta forma es posible elegir el candidato que obtiene mayor puntaje y asociar un nivel de confidencia a dicha percepción.

Como ya se mencionó, originalmente las características que determinan los puntajes se definían empíricamente, a partir de criterios heurísticos. Sin embargo este criterio puede ser subjetivo y no necesariamente se obtiene un comportamiento representativo.

A diferencia del sistema de percepción descrito anteriormente, se reemplazan en esta parte los criterios heurísticos por un clasificador, construido a partir de una red neuronal feed-forward, para determinar el puntaje de una región. El clasificador recibe como entradas un conjunto de características extraídas de cada candidato, por lo que es necesario realizar una etapa de pre-procesamiento de la imagen.

En la Figura 15 es posible apreciar la configuración del arco. Se propone como una buena caracterización, descomponer el arco en sus respectivos postes y travesaño de manera de poder identificarlos y conocer sus propiedades. De esta forma es posible obtener información relevante del arco y su posición.

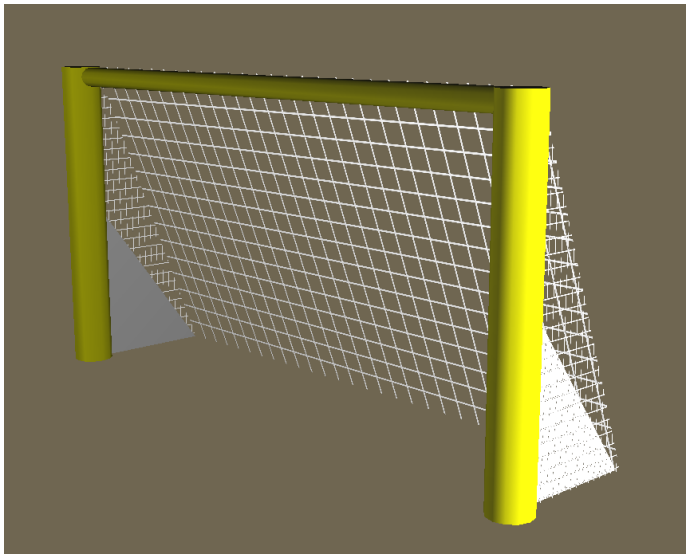

Figura 15: Configuración del arco.

A continuación se describe en detalle el trabajo realizado sobre el perceptor del arco.

3.2.1 Reglas Binarias

El objetivo de esta parte es filtrar regiones en la imagen que estén relacionadas con objetos relevantes en el campo de juego. El filtrado de objetos es realizado sometiendo un conjunto de regiones candidatas de la imagen (o combinaciones de ellas) a un set de reglas binarias. Luego las regiones que no cumplan con las condiciones mínimas de aceptación son eliminadas del conjunto de candidatos. Por ejemplo, la detección del arco usualmente requiere que la región relacionada tenga un tamaño mínimo y si no es el caso se espera que la región candidato sea rechazada.

3.2.2 Líneas de Escaneo y Extracción de Características

El objetivo en este punto es obtener la mayor cantidad de información posible de cada uno de los candidatos que hayan superado el filtrado por reglas binarias. Para esto se utilizan líneas de escaneo trazadas a partir de la información básica que se tiene de las regiones candidatas. Este ejercicio permite descomponer el arco en sus respectivos postes, siempre y cuando estos se encuentren dentro de los límites de la imagen. En la Figura 16 se muestra un ejemplo de una caracterización. A partir de las líneas de escaneo (perpendiculares a cada uno de los postes) se marcan los puntos de transición entre el arco y el resto de la imagen segmentada, los cuales son utilizados para ajustar una línea recta, por *Ransac – Random Sample Consesus*[4] (Muestra Aleatoria de Consenso), que determina la cara interna y externa de cada poste.

[4] El algoritmo *Ransac* es un método para ajustar un modelo de forma robusta en presencia de muchos datos anómalos.

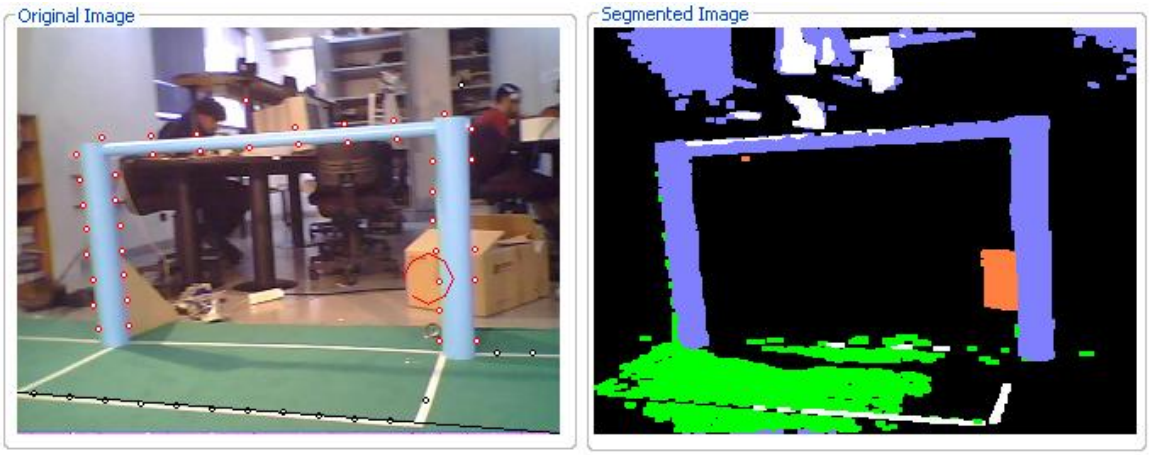

Figura 16: Líneas de escaneo.

Junto con la información obtenida del análisis de la región por líneas de escaneo y el procesamiento previo es posible extraer el siguiente listado de características, usado como entrada para la red neuronal.

Nombre	**Tipo**	**Descripción**
width	double	ancho
height	double	alto
size	int	tamaño en pixeles
areaBBX	double	área que envuelve al blob
ratio	double	proporción de color
isBorder	bool	indica si intersecta el borde de la imagen
aspectHV	double	relación entre el ancho y alto
simpleDist	double	distancia al arco
rectangleShape	double	relación entre mínimos ancho y alto, y areaBBX
totWidth	double	promedio entre ancho superior e inferior
totHeight	double	promedio entre alto izquierdo y derecho
rightTriangleShape	double	mide tendencia triangular hacia la derecha
leftTriangleShape	double	mide tendencia triangular hacia la izquierda
heightPostDet	bool [2]	indica si se conoce alto poste izquierdo/derecho
widthPostDet	bool [2]	indica si se conoce ancho poste izquierdo/derecho
crossBarDet	bool	indica si se detectó el travesaño

Tabla 1. Características extraídas de una región candidato.

3.2.3 Clasificador con Red Neuronal

La construcción del clasificador requiere resolver 3 etapas: creación de la base de datos, entrenamiento de la red e implementación de esta estructura en el código.

3.2.3.1 Creación de la base de datos

La base de datos se construye a partir de un conjunto de videos capturados por la cámara de los robots. Estos videos fueron tomados desde posiciones fijas y con el robot en movimiento, simulando situaciones reales de juego. Es necesario tener el objeto de interés, es decir el arco, contextualizado en diferentes situaciones para poder entrenar adecuadamente la red neuronal. Para esto se incorporan en la base de datos imágenes del arco desde distintas perspectivas: de frente y desde los costados, y a diferentes distancias. Además se incluyen secuencias de videos con el robot en movimiento, ya que simula el "*blur*" (figuras borrosas) asociado a las vibraciones del robot al moverse.

Una vez obtenidos los videos se realiza una etapa de extracción de características y etiquetado. Para esto existe una interfaz implementada en el código del equipo de fútbol robótico, donde es posible analizar los videos frame a frame y guardar las características de cada percepción indicando además si es correcta o no. Esta información es guardada en un archivo, que posteriormente es utilizado para el entrenamiento de la red.

Finalmente se obtiene una base de datos con más de 500 ejemplos de percepciones correctas y erróneas. Para entrenar la red se divide la base de datos en un conjunto de entrenamiento y uno de validación, dejando además un conjunto de prueba para evaluar la red.

3.2.3.2 Entrenamiento de la red

En una etapa previa al entrenamiento, la red es sometida a una optimización desde un punto de vista estructural. Es posible observar la configuración general de la red neuronal en la Figura 17:

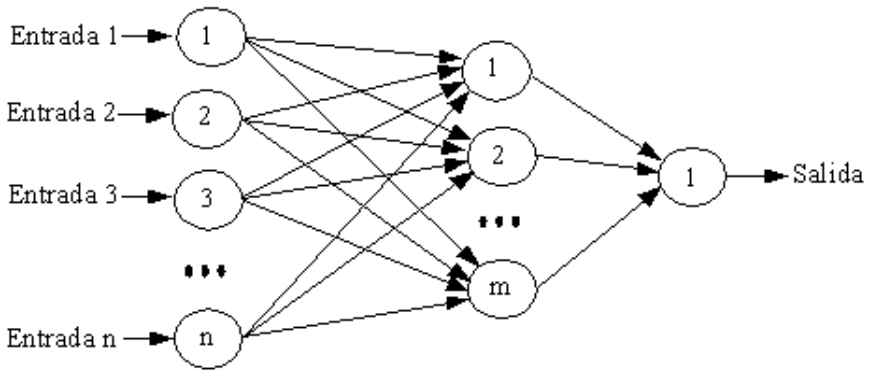

Figura 17. Estructura general de una red neuronal *MLP*.

donde *n* es el número de entradas a la red y *m* el número de unidades que conforman la capa intermedia u oculta. El número de entradas es fijo, y está definido por el tamaño del conjunto de características extraídas de una región. Por otro lado, entendiendo que se trata de un clasificador que debe entregar un puntaje a cada conjunto de entradas, la salida consiste en una única unidad. Luego, estructuralmente se desea encontrar el valor de *m* óptimo para este problema.

Para entrenar la red neuronal se utiliza el algoritmo *Back Propagation*, utilizando un conjunto de validación para evitar el sobre entrenamiento de la red. Para ajustar el número de unidades en la capa oculta se define en un comienzo una tasa de aprendizaje $\mu = 0{,}1$.

El número de unidades de la capa oculta se relaciona con la capacidad de aprendizaje de la red. Este parámetro se debe ajustar de acuerdo a la complejidad del problema. Se prueba el rendimiento de la red para distintas

cantidades de unidades en la capa oculta. Todos los resultados que se muestran a continuación se obtuvieron sobre el conjunto de prueba con los datos normalizados[5]. La normalización realizada para cada valor fue la siguiente:

$$x'_{ij} = \frac{x_{ij} - \mu_i}{\sigma_i}$$

con $\mu_i = \frac{\sum_j^N x_{ij}}{N}$ y $\sigma_i = \sqrt{\frac{\sum_j^N (x_{ij} - \mu_i)^2}{N}}$

donde μ_i corresponde a la media y σ_i a la desviación estándar de los datos. Se muestra a continuación el resultado para distintas configuraciones incluyendo además el valor del error cuadrático medio *mse*:

N° unidades capa oculta	Clasificaciones correctas [%]	mse
1	95,11	0,041
	93,71	0,051
	95,11	0,048
Promedio	94,64	0,047

Tabla 2. Rendimiento para m = 1

N° unidades capa oculta	Clasificaciones correctas [%]	mse
2	93,01	0,054
	94,41	0,048
	96,5	0,031
Promedio	94,64	0,041

Tabla 3. Rendimiento para m = 2

N° unidades capa oculta	Clasificaciones correctas [%]	mse
5	93,71	0,05
	95,11	0,038
	93.01	0,054
Promedio	93,94	0,047

Tabla 4. Rendimiento para m = 5

[5] En el caso de las variables booleanas no fue necesario normalizar.

N° unidades capa oculta	Clasificaciones correctas [%]	mse
10	97,2	0,043
	97,9	0,036
	97,2	0,036
Promedio	97,44	0,039

Tabla 5. Rendimiento para m = 10

N° unidades capa oculta	Clasificaciones correctas [%]	mse
15	97,9	0,028
	96,5	0,034
	97,2	0,033
Promedio	97,2	0,032

Tabla 6. Rendimiento para m = 15

N° unidades capa oculta	Clasificaciones correctas [%]	mse
20	97,2	0,04
	96,5	0,033
	97,2	0,035
Promedio	96,97	0,036

Tabla 7. Rendimiento para m = 20

N° unidades capa oculta	Clasificaciones correctas [%]	mse
25	95,8	0,031
	97,2	0,03
	97,2	0,035
Promedio	96,74	0,032

Tabla 8. Rendimiento para m = 25

Se puede ver que el mejor rendimiento de la red se obtiene para $m = 10$, donde se alcanza el 97,44 [%] de clasificaciones correctas. Finalmente la configuración definitiva de la red queda con 18 entradas, 10 unidades en la capa oculta y una unidad de salida.

En el algoritmo *Back Propagation*, la tasa de aprendizaje μ determina la velocidad con que se aprende. Así, al escoger una tasa muy baja se obtiene

un aprendizaje lento, y por el contrario una tasa de aprendizaje muy alta provoca que los pesos y la función objetivo diverjan, y por lo tanto no haya aprendizaje.

Se muestran enseguida los resultados de las pruebas realizadas sobre el conjunto de prueba para distintos valores de μ. Para cada valor se realizan 3 pruebas y luego se toma el promedio. Se incluye para cada intento el valor del error cuadrático medio *mse* y la curva de aprendizaje obtenida:

μ	Clasificaciones correctas [%]	mse
0,001	91,61	0,084
	84,62	0,135
	88,81	0,095
Promedio	88,345	0,105

Tabla 9. Rendimiento para μ = 0,001

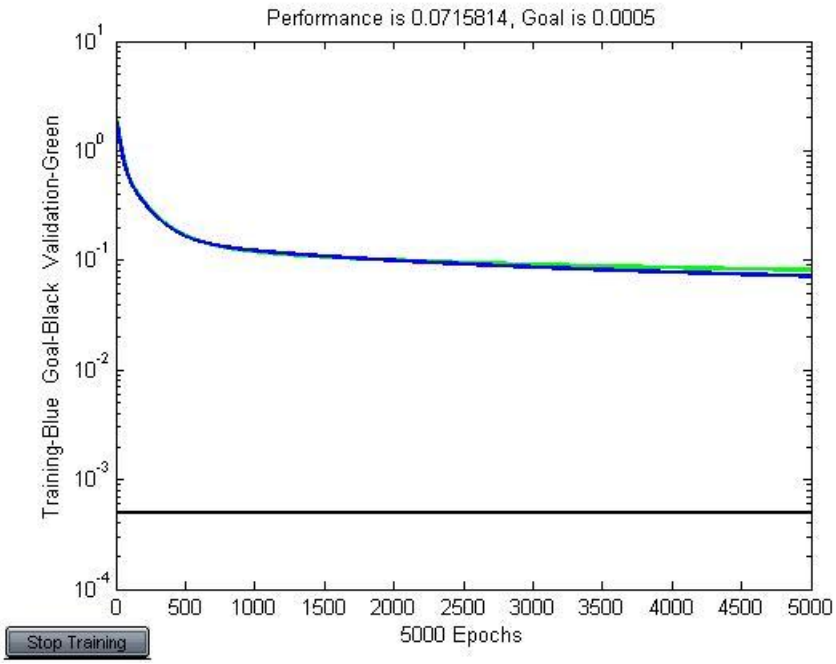

Figura 18. Curva de aprendizaje para μ = 0,001.

Como se ve en la Figura 18 el algoritmo presenta un aprendizaje relativamente lento, y se puede apreciar que la curva de entrenamiento se mantiene lejos del error objetivo.

Si se aumenta el valor de μ a 0,01 se obtiene:

μ	Clasificaciones correctas [%]	mse
0,01	90,91	0,056
	94,41	0,043
	92,31	0,061
Promedio	92,54	0,053

Tabla 10. Rendimiento para μ = 0,01

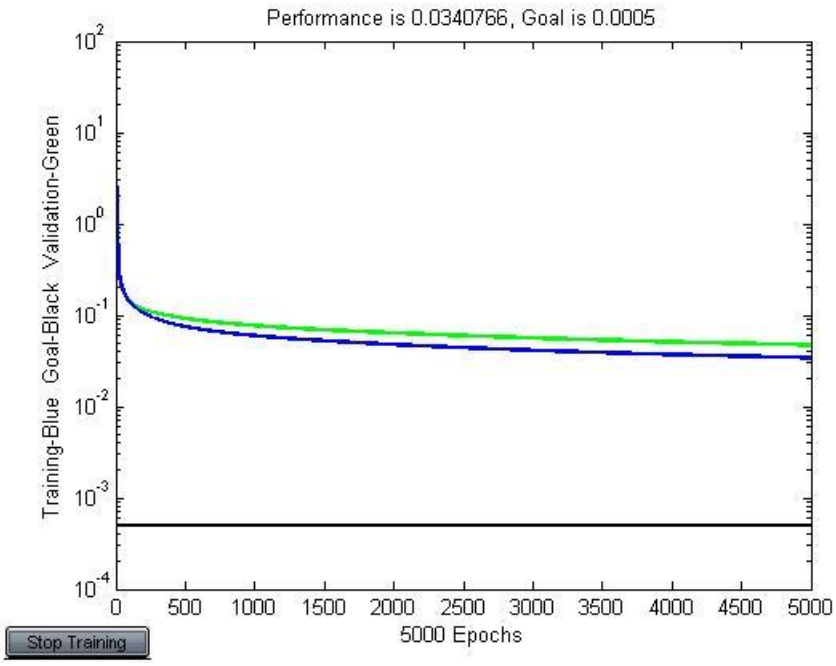

Figura 19. Curva de aprendizaje para μ = 0,01.

Para esta tasa se observa que el algoritmo converge más rápido. Sin embargo todos los entrenamientos se detienen por el límite de épocas y no por el conjunto de validación. Para $\mu = 0{,}1$:

μ	Clasificaciones correctas [%]	mse
0,1	97,9	0,033
	93,71	0,048
	96,5	0,036
Promedio	96,04	0,039

Tabla 11. Rendimiento para μ = 0,1

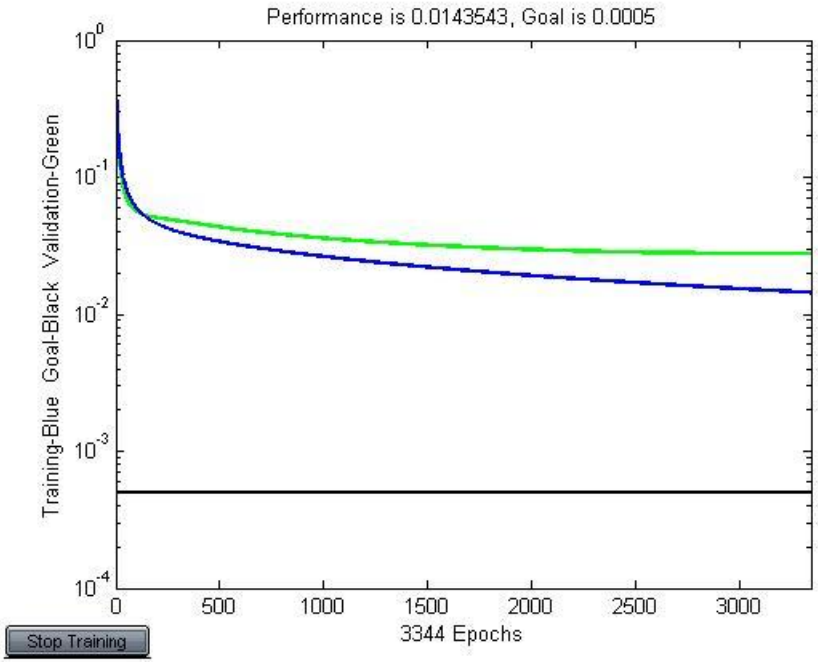

Figura 20. Curva de aprendizaje para μ = 0,1.

Con este valor se obtiene un porcentaje de clasificaciones correctas más alto que para los casos anteriores. Se observa además que la curva de aprendizaje converge más rápido y se detiene esta vez por el conjunto de validación.

Para tasas de aprendizaje más altas se obtiene:

μ	Clasificaciones correctas [%]	mse
0,15	97,9	0,032
	97,2	0,026
	97,9	0,033
Promedio	97,68	0,03

Tabla 12. Rendimiento para μ = 0,15

μ	Clasificaciones correctas [%]	mse
0,25	45,46	1,77
	57,34	1,42
	23,08	2,38
Promedio	41,96	1,86

Tabla 13. Rendimiento para μ = 0,25

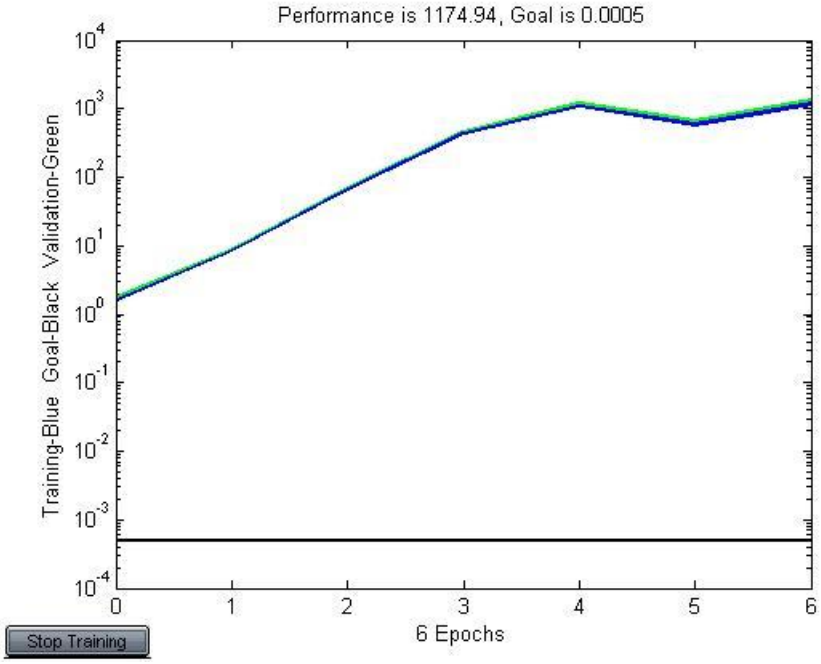

Figura 21. Curva de aprendizaje para μ = 0,25.

Como era de esperar para $\mu = 0{,}25$ el algoritmo diverge lo que se traduce en una interrupción temprana del entrenamiento y un pobre porcentaje de clasificaciones correctas.

A partir de los resultados obtenidos se elige $\mu = 0{,}15$ ya que entrega los mejores resultados de clasificación, los cuales se acercan al 98 [%]. Con esto se encuentra la configuración final de la red.

3.2.3.3 Implementación en el código

Para implementar esta estructura en el código se crea como clase principal `UChFeedForwardNeuralNetwork()`. Entre otras funciones, dentro de esta clase se pueden encontrar:

```
UChFeedForwardNeuralNetwork(UChDynamicArray<int> size);
```

Constructor de la clase que recibe como argumento un arreglo de enteros que determina el número de capas y su tamaño. Se inicializan por defecto todas las funciones de activación lineales.

```
void ReadWeights(FILE *fd);
```

Esta función lee los pesos de un archivo y los carga en la variable `weights` de la red neuronal.

```
UChVector EvalNetwork(const UChVector &input);
```

Función que evalúa la red a partir de un vector de entradas y entrega como salida el resultado de la red en un vector.

```
void SetActivationFunction(ActivationFunction newFunction,
int numLayer);
```

Esta función permite modificar las funciones de activación de una capa o de la red completa.

En la clase del perceptor del arco se crea una estructura de este tipo, en la cual se cargan los pesos desde un archivo predefinido y se utiliza para entregar un puntaje a las regiones candidatas a arco.

3.3 VISIÓN MULTI-RESOLUCIÓN

A pesar de los avances tecnológicos en componentes electrónicos, los sistemas montados actualmente en robots siguen siendo lentos y caros. Es por esto que la mayoría de los esfuerzos en visión computacional se han concentrado en el desarrollo de algoritmos de procesamiento cada vez más eficientes. Se propone en esta sección un

método basado en multi-resolución, que reduce los tiempos de procesamiento del módulo de visión.

Como es descrito en la sección 2.1.2, en la primera etapa del módulo de visión la imagen capturada por la cámara es segmentada, asociando cada pixel a una clase de color. Considerando las dimensiones de la imagen este procedimiento tiene un costo computacional alto: 320 x 240 = 76.800 pixeles analizados. Esto convierte al módulo de visión en una etapa crítica del sistema, con respecto a los tiempos de procesamiento. Con el objetivo de aminorar el número de operaciones realizadas, en esta sección se propone un sistema de escaneo de la imagen utilizando diferentes resoluciones de acuerdo a la importancia del contenido de cada zona: en otras palabras un algoritmo multi-resolución para segmentar la imagen. Desde este punto de vista, multi-resolución cobra interés ya que es un método eficiente para reducir la cantidad de cálculos.

A modo de aclaración es posible decir que en el área de procesamiento de imágenes existe más de una acepción para el término multi-resolución. Seguramente la más conocida tiene que ver con la técnica donde la imagen se representa en forma piramidal, para lo cual se aplica un filtro para pasar de un escalón a otro [10]. Sin embargo, en este trabajo se utiliza este término para denotar un algoritmo que reemplaza el método existente, en el cual se procesan todos los píxeles, por uno donde los píxeles que se procesan son muestreados en forma determinística y su frecuencia de muestreo está dada por la región de la imagen.

Para ejemplificar este procedimiento, en la Figura 22 se presenta el resultado de la segmentación tradicional de una imagen ejemplo, donde cada pixel es asignado a una clase de color, que para este caso son solo dos.

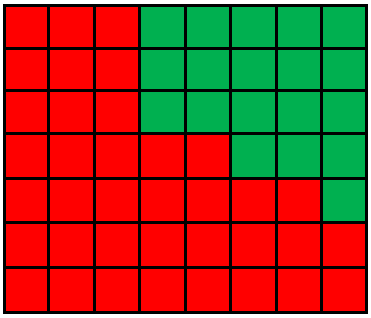

Figura 22. Imagen segmentada considerando todos los pixeles.

Como se aprecia, en la Figura 22 se utiliza una resolución de escaneo de 1x1, es decir se procesan todos los pixeles de la imagen.

Para resoluciones menores, es posible establecer un área dentro de la cual se escoge un representante único del sector completo. Dependiendo de la clasificación del color del pixel escogido se consideran todos los pixeles del área de la misma clase. La posición del pixel representante del área es arbitraria, pudiendo ser un pixel central o de alguna esquina.

En la Figura 23 se escanea la misma imagen mencionada arriba pero se utilizan distintas resoluciones y se define como el pixel representante de cada región el que está ubicado en la esquina superior izquierda. El resultado obtenido se puede apreciar en la Figura 24, donde se rellenan las áreas con el color de la clase del pixel escogido.

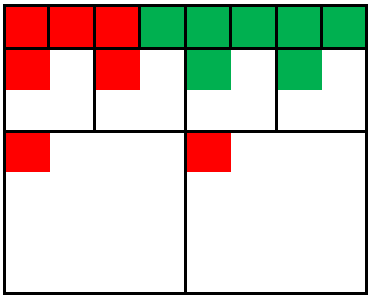

Figura 23. Pixeles representantes de áreas de tamaño: 1x1, 2x2 y 4x4.

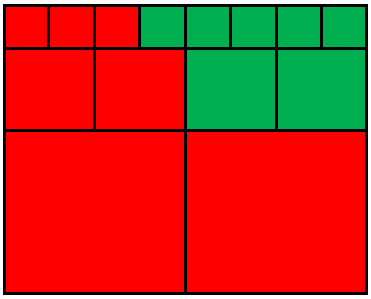

Figura 24. Imagen segmentada utilizando diferentes resoluciones.

Hasta acá resulta bastante sencillo. Sin embargo es necesario definir algún criterio razonable a partir del cual se determine el tamaño de cada área de escaneo dentro de la imagen. Dado que lejos del horizonte[6] solo es posible ver los objetos que están cerca del robot y que por lo tanto se ven más grandes y con más detalle, no es necesario procesar todos lo píxeles para detectarlos. Basándose en este hecho se definen las resoluciones de escaneo de la imagen en función de la distancia al horizonte. Donde claramente mientras más cerca se esté del horizonte, mayor será la resolución.

Específicamente se definen bandas paralelas al horizonte de un ancho fijo. Dentro de cada banda se utiliza una resolución única de escaneo. El ancho de cada franja es de 30 pixeles y las resoluciones comienzan en los alrededores del horizonte en cuadros de 1x1 y a medida que se alejan y en orden las resoluciones de cada franja son: 2x2, 4x4, 8x8 y la resolución mínima es de 16x16. Para definir estos parámetros se toman en cuenta dos factores: el tamaño de los objetos de interés; y la frecuencia con que un objeto es encontrado en zonas alejadas del horizonte.

Para el manejo de estas estructuras se crea una clase llamada `UChSegmentationZone()`, que dentro de sus atributos contiene un arreglo de segmentos run-length, un entero que guarda el punto de inicio y otro el de fin, que definen los límites de la zona y un entero que guarda su resolución.

---

[6] Se define el horizonte como la intersección entre el plano de la imagen y el plano paralelo al suelo ubicado a la altura de la cámara.

En este punto se introduce una simplificación importante: el horizonte se considera siempre horizontal dentro del plano de la imagen. Esta consideración simplifica bastante la implementación del algoritmo y es razonable ya que morfológicamente los robots no pueden girar el cuello en el plano de la imagen (ver Figura 25).

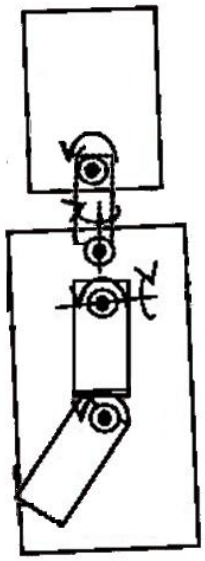

Figura 25. Grados de libertad del cuello de robot visto desde un costado.

Si bien es cierto, las oscilaciones propias de la caminata del robot se traducen en perturbaciones en la horizontalidad de la línea del horizonte, éstas son menores y es posible despreciarlas. Si las perturbaciones fueran mayores, y no fuera posible despreciarlas, se podrían usar bandas inclinadas, sin embargo el problema se complejizaría bastante.

En la etapa de implementación se crean funciones que permiten un recorrido más libre de la imagen. Originalmente, al escanear la imagen, el siguiente pixel analizado es siempre contiguo al actual. Luego para avanzar a través de la imagen se utiliza una función que simplemente actualiza la posición del puntero, incrementándola en uno. Al

introducir más de una resolución, se requiere contar con una función que permita saltar entre distintas posiciones de la imagen. Para esto se crea la función:

```
UChColorPixel GetImagePixel(int row, int col)
```

la cual recibe como argumentos una fila y columna cualquiera dentro de la imagen y entrega el pixel ubicado en esa posición.

La codificación run-length es realizada para cada zona por separado. Esto es necesario ya que al tener distintas resoluciones no es posible mezclar segmentos de diferentes tamaños. Los segmentos resultantes se guardan en los arreglos propios de cada zona.

Nuevamente dentro de cada zona se realiza la formación de regiones de manera similar al caso simple (ver Figura 26). Con esto se obtiene para cada color y cada resolución un arreglo de regiones caracterizadas por sus 8 puntos extremos, su centro de masa y su tamaño.

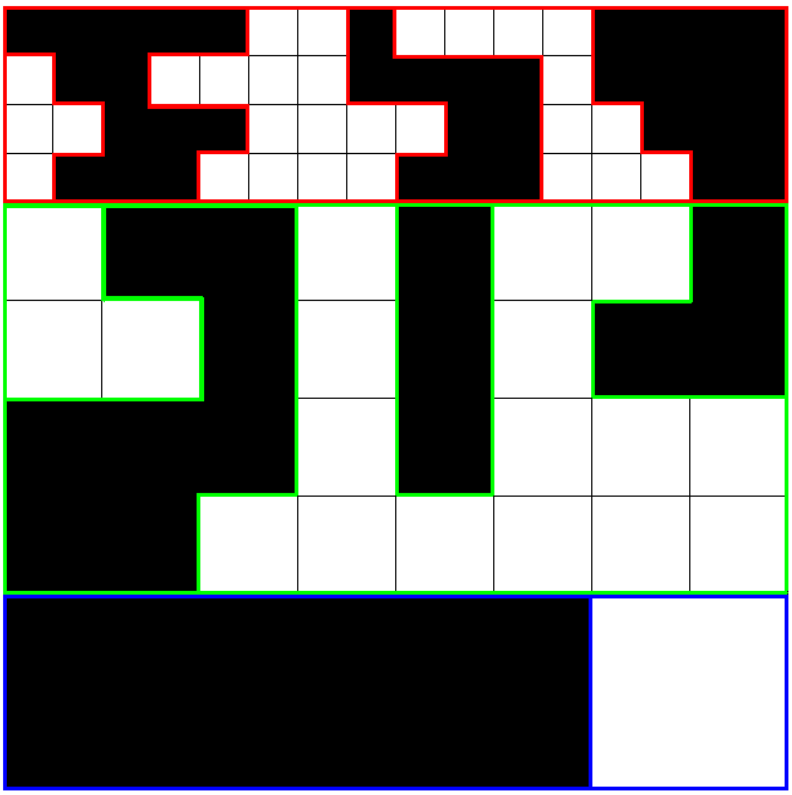

Figura 26. Ejemplo de segmentación por zonas.

Es necesario ahora combinar las regiones formadas a partir de distintas resoluciones de escaneo. Para esto se analizan las fronteras formadas entre zonas: si se encuentran dos regiones colindantes del mismo color se unen, es decir, se elimina una de las zonas y se actualiza la información de la otra considerando las características de ambas (ver Figura 27). Los blobs resultantes se copian en un nuevo arreglo junto con aquellos que no fue necesario modificar.

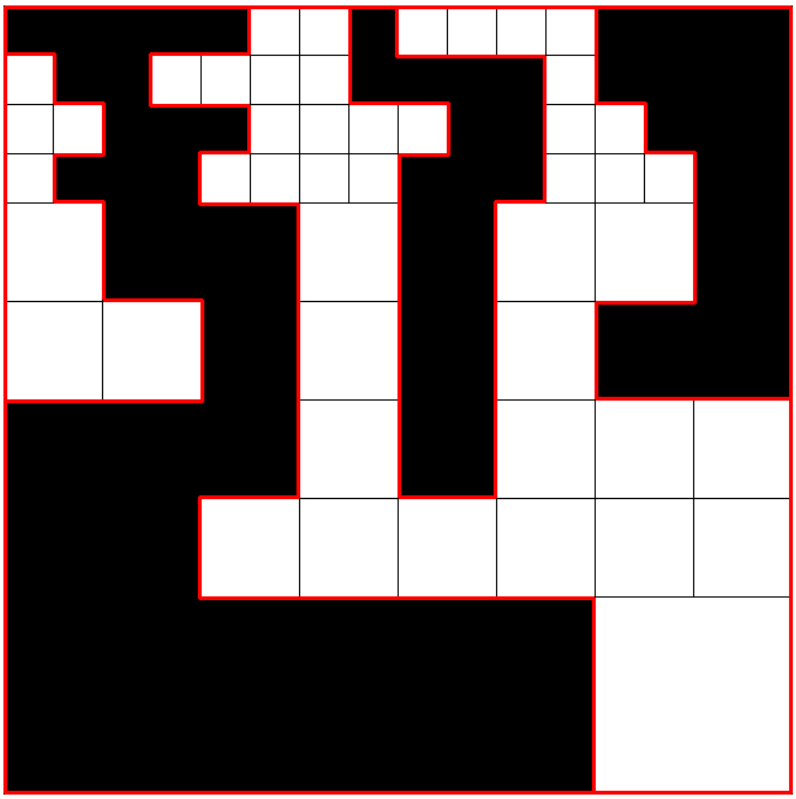

Figura 27. Segmentación final.

De aquí en adelante se utiliza la estructura del código original.

IV. PRUEBAS Y ANÁLISIS

4.1 Resultados Obtenidos

En esta sección se presentan los resultados obtenidos.

4.1.1 Precisión de Mapeo

El lente gran angular usado para los experimentos es fabricado por MXC y su distancia focal es 2,5 [mm]. El ángulo de visión es de 130 [º]. El lente fue montado sobre una cámara CCD marca Philips con una resolución interpolada de 1,3 MP.

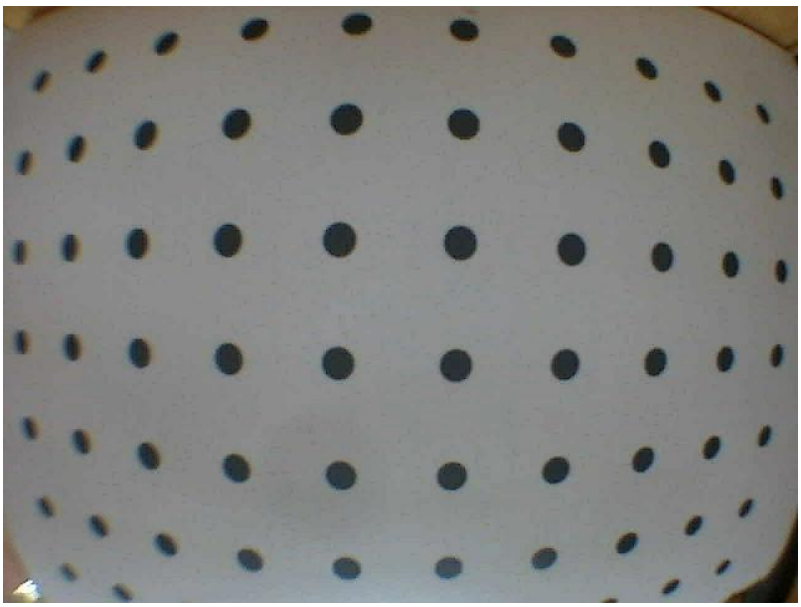

Figura 28. Patrón de calibración visto por lente gran angular.

La Figura 28 muestra una imagen de 640 x 480 de una grilla de puntos co-lineales tomada con el lente gran angular. Es posible ver una curvatura en la estructura de puntos producto de la distorsión introducida por el lente. Esta imagen fue corregida usando la transformación polinomial. La imagen corregida se puede ver en la Figura

29, donde cada pixel en la imagen distorsionada es corregido y mapeado en la nueva imagen. Cuando se corrige, la imagen generada es de varias veces el tamaño de la imagen original. En la Figura 29 el tamaño es restringido a una resolución de 662 x 535 pixeles. Como se ve, hay muchos pixeles en negro presentes en la imagen corregida. Esto se debe al efecto que produce el mapear una imagen de 640 x 480 en una imagen más grande. Sin embrago esto no es relevante para la aplicación en el sistema de visión.

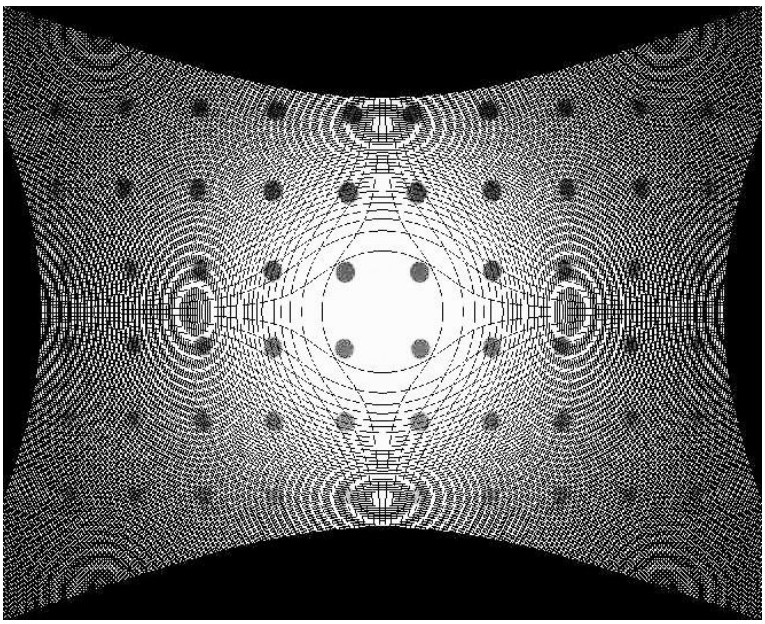

Figura 29. Imagen corregida.

Se puede ver en la Figura 29 que la curvatura de los puntos es prácticamente imperceptible, corrigiéndose con esto la distorsión del lente. Es importante recalcar que las dimensiones de los objetos en la imagen corregida no cambian, si se comparan con los tamaños de los mismos objetos capturados con un lente de proyección lineal desde la misma posición, y se conservan las proporciones de los tamaños dentro de la imagen.

Para tener una medida cuantitativa de la precisión del mapeo es posible realizar un ajuste de mínimos cuadrados para las filas y columnas de los puntos resultantes.

Este procedimiento arroja para cada ajuste un vector de residuos $\vec{r}_i$, de tamaño igual al número de puntos que se consideran. Luego para medir la precisión se puede considerar el siguiente indicador p:

$$p = \max_i \{\|\vec{r}_i\|\}$$

es decir el máximo de las normas de los residuos obtenidos. Al realizar este cálculo para la imagen corregida se obtiene que $p = 1{,}02$.

Para tener una referencia de la precisión lograda, es posible comentar que el procedimiento de búsqueda de los coeficientes de distorsión del polinomio originalmente era realizado a mano. Se hacían pruebas para distintos conjuntos de parámetros y utilizando un criterio visual se modificaban hasta encontrar un mapeo aceptable. Además de ser un procedimiento tedioso, se calculó la precisión p al mejor de los resultados, obteniéndose un valor de $p = 6{,}23$.

4.1.2 Percepciones del arco

En Figura 1 (Pág. 3), donde se diagrama el funcionamiento general de la arquitectura del software, se puede ver que la salida del sistema de visión se conecta con dos módulos importantes: localización y estrategia. A grandes rasgos, visión envía información sobre las percepciones de objetos y su respectiva confidencia. Dada esta estructura, la comunicación de percepciones erróneas por parte de visión es nociva para el sistema completo. Es natural entonces medir el rendimiento del perceptor del arco a partir del porcentaje de clasificaciones correctas y la tasa de falsos positivos.

Se entrena la red considerando la mejor configuración encontrada en la sección 3.2.3, se fija el umbral de salida en 0,5 y se utiliza el conjunto de prueba, con lo cual se obtienen los siguientes resultados:

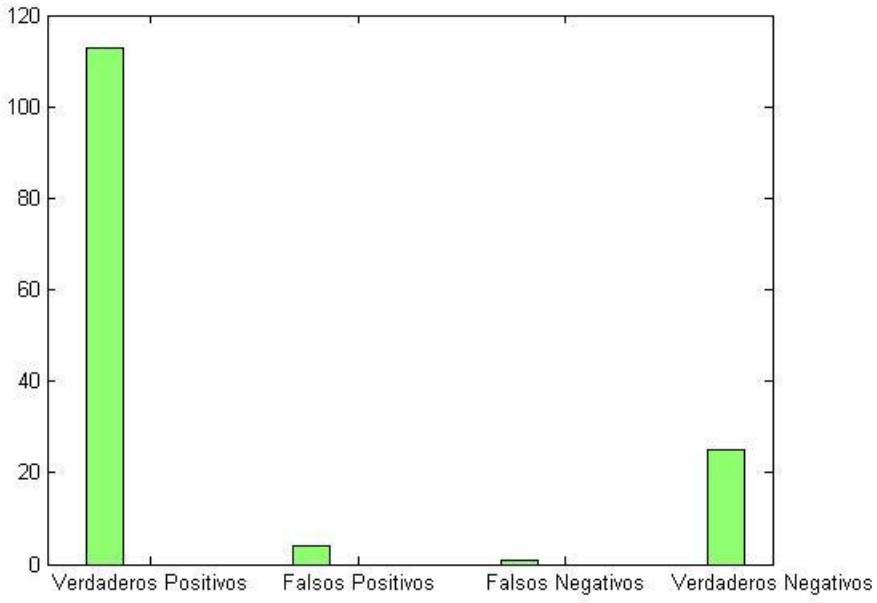

Figura 30. Rendimiento de la red para su mejor configuración.

Se puede apreciar en la Figura 30 que se logra un bajo número de falsos positivos. Este es un resultado positivo ya que la información del arco es necesaria no solo para definir la estrategia de juego, sino que además se utiliza en el módulo de localización del robot.

Una variable que influye fuertemente en la percepción de objetos es la calidad de la segmentación de la imagen, proceso que como se dijo, se lleva a cabo antes de detectar los objetos de interés. Esto se vuelve aun más relevante cuando la tarea consiste en detectar objetos donde es fácil perder información producto de una mala segmentación.

A continuación se realizan pruebas donde para un conjunto de videos se construyen distintas tablas de colores y la red se entrena para cada una de las segmentaciones:

N° experimento	% Aciertos	% Falsos positivos
1	96,5	2
2	95,8	2,5
3	97,1	0,4
4	96,3	1,5
5	96,4	1,7
6	96,8	1,1
Promedio	96,5	1,5

Tabla 14. Rendimiento.

Como se ve en la Tabla 14, al entrenar la red para distintas tablas de colores se obtienen resultados de clasificación similares y satisfactorios.

Si por el contrario, se usa una única tabla de colores para todos los videos y el clasificador se entrena una sola vez para esta tabla se obtienen los siguientes resultados:

N° experimento	% Aciertos	% Falsos positivos
1	96,5	2
2	86,4	7,2
3	84,1	9
4	73,9	13
5	87,4	9,1
6	80,8	10,8
Promedio	84,9	8,5

Tabla 15. Rendimiento.

Para esta prueba (Tabla 15) no se obtienen muy buenos resultados. El porcentaje de clasificaciones correctas depende fuertemente de la calibración de colores. Es claro entonces que es necesario adaptar la tabla de colores a las condiciones de iluminación ambientales para obtener buenos resultados con la red.

4.1.3 Recorrido de la imagen en multi-resolución

En el caso tradicional se recorren todos los pixeles de la imagen al momento de la segmentación. Dado que la imagen utilizada tiene una resolución de 320x240 el número total es pixeles procesados es de

$$N_{trad} = 320 \cdot 240 = 76.800$$

Ahora bien, si se recorre la imagen con distintas resoluciones y tomando como criterio de importancia la distancia del horizonte, se tiene que el peor de los casos se da cuando el horizonte se encuentra justo en centro de la imagen. Esto es fácil verlo en la Figura 31:

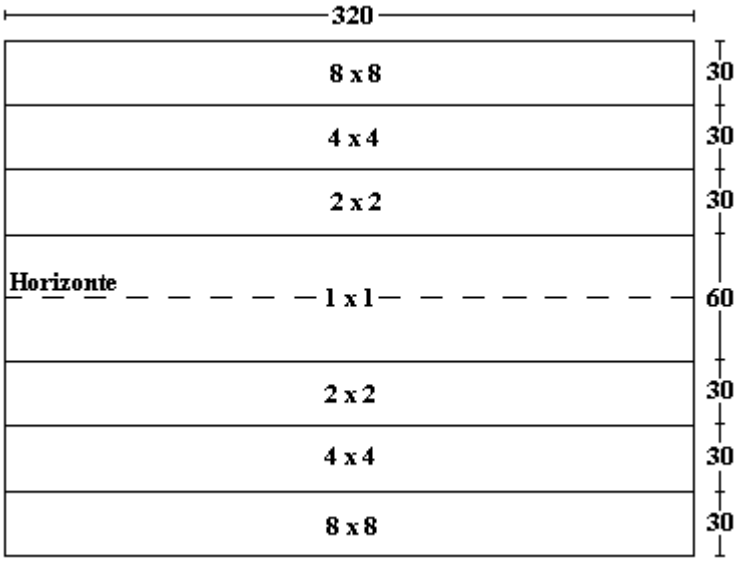

Figura 31. Peor caso de análisis en multi-resolución.

El número total de píxeles recorridos se puede aproximar[7] por

$$N_{multi} = 320 \cdot 60 \left(\frac{1}{64} + \frac{1}{16} + \frac{1}{4} + 1 \right) = 25.500$$

Se puede apreciar que la cantidad de cálculos se reduce a un tercio, reduciendo los tiempos de procesamiento de la imagen considerablemente.

Este es un resultado importante ya que las consideraciones de diseño de este algoritmo aseguran que la cantidad de información relevante perdida es baja y se logra un ahorro de cálculos significativo.

4.2 TRABAJO FUTURO

En el diseño de sistemas robóticos una de las premisas más importantes es tratar siempre de obtener el mejor desempeño del hardware disponible a través de una programación inteligente del software. A su vez, lo sofisticado de los algoritmos debe adecuarse a la capacidad de los componentes electrónicos que se disponen. A pesar de esto, la evolución de la tecnología permite implementar sistemas cada vez más complejos, capaces de resolver tareas más específicas. Por este motivo, y sobre todo en el contexto de este trabajo, es razonable considerar mejoras en los algoritmos, que a pesar de ser más complejas y "caras" computacionalmente, apuntan a resolver un determinado problema de forma más precisa dentro de un eventual escenario futuro.

Las correcciones del lente gran angular en este trabajo consideran solamente las distorsiones radiales. Sin embrago una aproximación más exacta puede tomar en cuenta también la distorsión tangencial que introduce el lente. Para esto es necesario utilizar dos polinomios, uno para corregir las variaciones radiales y otro para las variaciones angulares. Las ecuaciones generales para este caso son:

$$\rho' = a \cdot \rho^4 + b \cdot \rho^3 + c \cdot \rho^2 + d \cdot \rho$$

[7] Se entiende como un número aproximado ya que por motivos de eficiencia las franjas no necesariamente tienen un ancho de 30 pixeles. Sin embrago son siempre valores cercanos a este número.

$$\theta' = e \cdot \theta^4 + f \cdot \theta^3 + g \cdot \theta^2 + h \cdot \theta$$

donde la primera ecuación es idéntica a la utilizada anteriormente, y en la segunda el ángulo θ es definido como el ángulo entre la línea de proyección de un punto en el plano de la imagen y el centro óptico del plano de la imagen. Los coeficientes de distorsión quedan definidos por las variables *a* hasta *h*, y nuevamente pueden ser encontrados utilizando el algoritmo de optimización *PSO*. Si bien es cierto este enfoque no introduce mayores beneficios para el problema propuesto en este trabajo, puede ser útil para aplicaciones donde la precisión del mapeo es un parámetro más relevante.

En el caso de método implementado para el recorrido de la imagen con múltiples resoluciones no deja de ser interesante el paso de eliminar la suposición de horizontalidad de las franjas. De esta forma es posible generalizar el método para otro tipo de arquitecturas de robots (como los *AIBO*), con lo que se elimina esta limitante de hardware.

Existen diversas maneras de abordar el problema de multi-resolución. Siguiendo la misma línea de este trabajo, una variación del criterio de importancia de las zonas para definir resoluciones más finas puede ser: recordar la posición de los objetos de interés determinada en percepciones anteriores, para luego buscar con más detalle en las zonas de la imagen donde deberían encontrarse estos objetos.

V. CONCLUSIONES

Los resultados obtenidos muestran que los métodos propuestos tienen un buen desempeño. En cada etapa se aprecia la ventaja de su uso, por lo que se considera el trabajo realizado como una mejora integral al sistema de visión del equipo de fútbol robótico.

Se presenta en esta memoria un procedimiento efectivo para eliminar la distorsión introducida en una imagen producto del uso de un lente gran angular. En este proceso se usa una transformación polinomial para corregir la distorsión radial del lente. El polinomio es calculado usando un algoritmo de optimización *PSO* y se utiliza un patrón de calibración para medir la distorsión introducida. El procedimiento de corrección se aplica satisfactoriamente en imágenes capturadas a través del lente gran angular, y es posible medir la precisión de la corrección realizada. Dado que la transformación se aplica solo a los objetos de interés en el campo de juego, no tiene un costo computacional alto y puede ser aplicada en tiempo real.

Se considera suficiente la precisión obtenida a partir de este método para la aplicación deseada. Queda sin embrago la alternativa de lograr mejores correcciones sin modificar mayormente el método descrito en este trabajo, como por ejemplo considerando las distorsiones tangenciales introducidas por un lente gran angular.

En el método de percepción del arco se implementa en su segunda etapa un clasificador basado en una red neuronal feed-forward. Recibe como entradas un conjunto de características extraídas de las regiones que satisfacen previamente un filtro de reglas binarias, y entrega como salida un puntaje que permite elegir la región que corresponde a un arco. Los resultados de clasificación obtenidos son satisfactorios, obteniéndose un alto porcentaje de clasificaciones correctas y un bajo número de falsos positivos.

Por otro lado la introducción del clasificador no afecta la eficiencia del módulo de percepción, ya que a pesar de ser una solución compleja desde el punto de vista de su construcción, la implementación resulta bastante sencilla y de rápida evaluación.

Es importante notar que la red neuronal no soluciona el problema de segmentación, ya que como se aprecia en los resultados obtenidos, una mala segmentación se traduce en una baja calidad en las percepciones obtenidas.

Se implementa además un método de recorrido de la imagen, que a partir de un criterio determinado a priori, utiliza distintas resoluciones de escaneo. El criterio utilizado es que en la mayoría de los casos los objetos de interés se encuentran cercanos al horizonte. Luego se utilizan mayores resoluciones para zonas cercanas al horizonte y menores para zonas alejadas de éste. Esto permite reducir considerablemente el número de cálculos y aliviana así el proceso más pesado del sistema de visión.

VI. BIBLIOGRAFÍA

[1] C. Beck. “Apparatus to photograph the whole sky”. *Journal of Scientific Instrumentation, Vol. 2, pp. 135-139*, 1925.

[2] K. Miyamoto. “Fish eye lens”. *Journal Letter, Vol. 54, pp. 1060-1061*, 1964.

[3] R. L. Anderson, N. Alvertos, E.L. Hall. “Omnidirectional real time imaging using restoration”. *SPIE High Speed Photography, Vol. 348*, 1982.

[4] S. Shah, J.K. Aggarwal. “A simple calibration procedure for fish-eye (high distortion) lenscamera”. *IEEE International Conference on Digital Object Identifier, Vol. 4, pp. 3422-3427*, 1994.

[5] J.C. Zagal, J. Ruiz-del-Solar, P. Guerrero, R. Palma. “Evolving Visual Object Recognition for Legged Robots”. *LNAI Proceedings of RoboCup 2003: Robot Soccer World Cup VII*, Springer, 2003.

[6] X. Xie, W. Zhang Z., Yang. “Solving Numerical Optimization Problems by Simulating Particles in Potential Fields with Cooperative Agents”. *International Conference on Artificial Intelligence*, Las Vegas, NV, USA, 2002.

[7] G. Gudise, G. Venayagamoorthy. “Comparison of Particle Swarm Optimization and Backpropagation as Training Algorithms for Neural Networks”. *Proceedings of the IEEE Swarm Intelligence Symposium 2003, pp. 110-117*, Indianapolis, Indiana, USA, 2003.

[8] K. Parsopoulus, E. Papageorgiou, P. Groumpos, M. Vrahatis. “A First Study of Fuzzy Cognitive Maps Learning Using Particle Swarm Optimization”. *Proceedings of the IEEE Congress on Evolutionary Computation 2003, pp. 1440-1447*, Canbella, Australia, 2003.

[9] N. Otsu. "A threshold selection method from gray-level histograms". *IEEE Trans. Sys., Man., Cyber. pp*. 62–66. 1979.

[10] M.G.P. Bartholomeus, B.J.A. Krose and A.J. Noest. “A robust multi-resolution vision system for target tracking with a moving camera”.

VII. APÉNDICE. CÓDIGO FUENTE DE LAS CLASES PRINCIPALES

LENTE GRAN ANGULAR

R.m

```
function [fxmin, xmin, Swarm, history]=R(imag)

global Pe;
global N;
global M;

nParam = 100;

for i=1:nParam
    param(i,:) = [rand rand rand rand];
end

RGB = imread(imag);

BW = Binarizar(RGB);

[N,M] = size(BW);

[L,num] = Regiones(BW);

P = Pos(L,num);

Pe = Etiquetar(P);

[fxmin, xmin, Swarm, history] = algPSO;
```

Binarizar.m

```
function BW = Binarizar(RGB)

I  = rgb2gray(RGB);

[n,m] = size(I);

level = graythresh(I);

BW = im2bw(I, level);
```

Regiones.m

```
function [L,num]=Regiones(BW)

[L,num] = bwlabel(BW,4);
```

Pos.m

```
function P = Pos(L,num)

for i = 1:num
   [r,c] = find(L==i);
   rc = [r c];
   [n,m] = size(rc);

   x = 0;
   y = 0;
   for j = 1:n
      x = x+rc(j,1);
      y = y+rc(j,2);

   end

   x = x/n;
   y = y/n;

   P(i,:) = [x y 0 0];

end
```

Etiquetar.m

```
function Pf = Etiquetar(Po);

Pf = Po;

[n,m] = size(Pf);

et = 0;
fila = 0;
columna = 0;

while(et == 0)

    minDistIzq = 1000;
    minDistDer = 1000;
    minDistUp = 1000;
    minDistLo = 1000;

    ptoIzq = 0;
    ptoDer = 0;
    ptoUp = 0;
```

```
    ptoLo = 0;

    for i=1:n
    if (((Pf(i,3)~=0 && Pf(i,4)==0)) || ((Pf(i,3)==0 &&
Pf(i,4)~=0)) || ((Pf(i,3)==0 && Pf(i,4)==0)))
        p = [Pf(i,1) Pf(i,2)];
        ptoAct = i;
        break;

    end
    end

    for i=1:n
        dist = sqrt((p(1)-Pf(i,1))^2+(p(2)-Pf(i,2))^2);
        ang = atand((Pf(i,2)-p(2))/(Pf(i,1)-p(1)));

        if (abs(ang)<20 && Pf(i,1)<p(1) && dist<minDistIzq)
            minDistIzq = dist;
            ptoIzq = i;
        end
        if (abs(ang)<20 && Pf(i,1)>p(1) && dist<minDistDer)
            minDistDer = dist;
            ptoDer = i;
        end
        if (90-abs(ang)<20 && Pf(i,2)>p(2) && dist<minDistUp)
            minDistUp = dist;
            ptoUp = i;
        end
        if (90-abs(ang)<20 && Pf(i,2)<p(2) && dist<minDistLo)
            minDistLo = dist;
            ptoLo = i;
        end

    end

    if (ptoIzq ~=0 && ptoDer ~=0)

        if (Pf(ptoIzq,3)==0 && Pf(ptoDer,3)==0)
            fila = fila+1;
            Pf(ptoAct,3) = fila;
            Pf(ptoIzq,3) = fila;
            Pf(ptoDer,3) = fila;

        end
        if (Pf(ptoIzq,3)~=0 && Pf(ptoDer,3)==0)
            Pf(ptoAct,3) = Pf(ptoIzq,3);
            Pf(ptoDer,3) = Pf(ptoIzq,3);
        end
        if (Pf(ptoIzq,3)==0 && Pf(ptoDer,3)~=0)
            Pf(ptoAct,3) = Pf(ptoDer,3);
            Pf(ptoIzq,3) = Pf(ptoDer,3);
        end
        if (Pf(ptoIzq,3)==Pf(ptoDer,3))
            Pf(ptoAct,3) = Pf(ptoDer,3);
        end
```

```
    if (Pf(ptoIzq,3)~=Pf(ptoDer,3))
        pCorr = max(Pf(ptoIzq,3),Pf(ptoDer,3));
        for t=1:n
            if(Pf(t,3)==pCorr)
                Pf(t,3) = min(Pf(ptoIzq,3),Pf(ptoDer,3));
            end

        end
    end

end
if (ptoIzq ==0 && ptoDer~=0)
    if (Pf(ptoDer,3)==0)
        fila = fila+1;
        Pf(ptoAct,3) = fila;
        Pf(ptoDer,3) = fila;
    else
        Pf(ptoAct,3) = Pf(ptoDer,3);
    end

end
if (ptoDer ==0 && ptoIzq~=0)
    if (Pf(ptoIzq,3)==0)
        fila = fila+1;
        Pf(ptoAct,3) = fila;
        Pf(ptoIzq,3) = fila;
    else
        Pf(ptoAct,3) = Pf(ptoIzq,3);
    end

end

if (ptoUp ~=0 && ptoLo ~=0)

    if (Pf(ptoUp,4)==0 && Pf(ptoLo,4)==0)
        columna = columna+1;
        Pf(ptoAct,4) = columna;
        Pf(ptoUp,4) = columna;
        Pf(ptoLo,4) = columna;

    end
    if (Pf(ptoUp,4)~=0 && Pf(ptoLo,4)==0)
        Pf(ptoAct,4) = Pf(ptoUp,4);
        Pf(ptoLo,4) = Pf(ptoUp,4);
    end
    if (Pf(ptoUp,4)==0 && Pf(ptoLo,4)~=0)
        Pf(ptoAct,4) = Pf(ptoLo,4);
        Pf(ptoUp,4) = Pf(ptoLo,4);
    end
    if (Pf(ptoUp,4)==Pf(ptoLo,4))
        Pf(ptoAct,4) = Pf(ptoLo,4);
    end
    if (Pf(ptoUp,4)~=Pf(ptoLo,4))
        pCorr = max(Pf(ptoUp,4),Pf(ptoLo,4));
        for t=1:n
            if(Pf(t,4)==pCorr)
```

```
                        Pf(t,4) = min(Pf(ptoUp,4),Pf(ptoLo,4));
                    end

                end
            end

        end
        if (ptoUp ==0 && ptoLo~=0)
            if (Pf(ptoLo,4)==0)
                columna = columna+1;
                Pf(ptoAct,4) = columna;
                Pf(ptoLo,4) = columna;
            else
                Pf(ptoAct,4) = Pf(ptoLo,4);
            end

        end
        if (ptoLo ==0 && ptoUp~=0)
            if (Pf(ptoUp,4)==0)
                columna = columna+1;
                Pf(ptoAct,4) = columna;
                Pf(ptoUp,4) = columna;
            else
                Pf(ptoAct,4) = Pf(ptoUp,4);
            end

        end


        et = EtListo(Pf);

    end
```

EtListo.m

```
    function l = EtListo(P)

    [n,m] = size(P);

    l = 1;

    for i=1:n
        if (P(i,3)==0 || P(i,4)==0)
            l = 0;
            break;
        end
    end
```

algPSO.m

```
function [fxmin, xmin, Swarm, history] = algPSO

psoOptions = get_psoOptions;

% PSO VARIABLES
psoOptions.Vars.SwarmSize = 100;
psoOptions.Vars.Iterations = 100;
psoOptions.Vars.ErrGoal = 1;
psoOptions.Vars.Dim = 3;

% OBJECTIVE FUNCTION OPTIONS
psoOptions.Obj.f2eval = 'psoObj';
psoOptions.Obj.lb = 0;
psoOptions.Obj.ub = 0.0000001;

% TERMINATION OPTIONS
psoOptions.Terminate.Iters = 1;
psoOptions.Terminate.Err = 1;

% STRATERGY PARAMETERS
psoOptions.Sparams.c2 = 4;
psoOptions.Sparams.w_start = 0.7;
psoOptions.Sparams.w_end = 0.3;
psoOptions.Sparams.w_varyfor = 0.0005;
psoOptions.Sparams.Vmax = 0.1;

% TEXT MODE DISPLAY OPTIONS
%psoOptions.Disp.Interval = 10;

[fxmin, xmin, Swarm, history] = pso(psoOptions)
```

psoObj.m

```
function ObjV = psoObj(swarm)

global Pe;
global N;
global M;

[n,m] = size(swarm);

for i=1:n

    Pm = Mapeo(Pe,N,M,swarm(i,:));
    ObjV(i) = Ajuste(Pm);

end
ObjV = ObjV';
```

Mapeo.m

```
function Pf = Mapeo(Po,N,M,param)

Pf = Po;
[n,m] = size(Po);

a = param(1);
b = param(2);
c = param(3);
d = 1;

for i=1:n
    x = Po(i,1) - round(N/2);
    y = Po(i,2) - round(M/2);
    [theta,r] = cart2pol(x,y);
    s = (a*r^3+b*r^2+c*r+d)*r;
    [Pf(i,1),Pf(i,2)] = pol2cart(theta,s);

end
```

Ajuste.m

```
function eMax = Ajuste(Pt)

[n,m] = size(Pt);
d = 1;
listo = true;
e = 0;
eMax = 0;

while (listo)
    k = 0;
    listo = false;

    for i=1:n
       if (Pt(i,3)~=0)
           idx = Pt(i,3);
           listo = true;
           break;
       end

    end
    if (listo)
        for i=1:n
           if (Pt(i,3)==idx)
              Pt(i,3) = 0;
              k = k+1;
              pto(k,:) = [Pt(i,1) Pt(i,2)];
           end

        end

        [p,S] = polyfit(pto(1:k,1),pto(1:k,2),d);
        e = S.normr;
```

```
            if (e>eMax)
                eMax = e;
            end

        end

    end
```

CLASIFICADOR RED NEURONAL

```
class UChFeedForwardNeuralNetwork

{
public:
    UChFeedForwardNeuralNetwork();
    // Constructor que recibe como parámetros el número de capas y sus
    // tamaños.
    // Setea por default todas la funciones de activacion lineales.
    UChFeedForwardNeuralNetwork(UChDynamicArray<int> size);

    ~UChFeedForwardNeuralNetwork();

    // Funciones

    // Setea la red
    void SetNetwork(UChDynamicArray<int> size);
    // Lee los pesos desde un archivo y los carga en weights
    void ReadWeights(FILE *fd);
    // Evalua la red a partir de un vector de entrada
    UChVector EvalNetwork(const UChVector &input);
    // Pasa las salidas lineales de una capa por la funcion de
    //activacion
    void ActivateLayer(int layerIdx);
    // Setea todas las funciones de activacion de la red
    void SetActivationFunction(ActivationFunction newFunction);
    // Setea la funcion de activacion de la capa numLayer
    void SetActivationFunction(ActivationFunction newFunction, int
    numLayer);

    UChDynamicArray<UChMatrix> weights; // Pesos de la red


private:
    int nLayers; // Numero de capas
    UChDynamicArray<int> layerSize; // Tamaño de las capas
    UChDynamicArray<ActivationFunction> activationFunctions; // Arreglo
    //con las funciones de activacion de las capas
    //UChDynamicArray<UChMatrix> weights; // Pesos de la red
    UChDynamicArray<UChVector> layerOutput; // Salida lineal de cada
    //capa
    UChDynamicArray<UChVector> activatedLayerOutput; // Salida pasada
    //por la capa de activacion

};
```

```
UChFeedForwardNeuralNetwork::UChFeedForwardNeuralNetwork(UChDynamicArray<
int> size)
{
	nLayers = size.getLen();
	layerSize = size;

	for (int i=0; i<nLayers-1; i++)
	{
		weights.push(UChMatrix(size[i+1],size[i]));

	}
	for (int i=0; i<nLayers; i++)
	{
		activationFunctions.push(LINEAR);
		layerOutput.push(UChVector(size[i]));
		activatedLayerOutput.push(UChVector(size[i]));
	}
}

UChFeedForwardNeuralNetwork::~UChFeedForwardNeuralNetwork()
{
}

void UChFeedForwardNeuralNetwork::SetNetwork(UChDynamicArray<int> size)
{
	nLayers = size.getLen();
	layerSize = size;

	for (int i=0; i<nLayers-1; i++)
	{
		weights.push(UChMatrix(size[i+1],size[i]));

	}
	for (int i=0; i<nLayers; i++)
	{
		activationFunctions.push(LINEAR);
		layerOutput.push(UChVector(size[i]));
		activatedLayerOutput.push(UChVector(size[i]));
	}
}

void UChFeedForwardNeuralNetwork::ReadWeights(FILE *fd)
{
	int height, width;
	double aux;

	for (int i=0; i<nLayers; i++)
	{
		height = weights[i].Height();
		width = weights[i].Width();
		for(int j=0; j<height; j++)
			for (int k=0; k<width; k++)
			{
				fscanf(fd,"%lf",&aux);
				weights[i][j][k] = aux;
			}
	}
```

```
}

UChVector UChFeedForwardNeuralNetwork::EvalNetwork(const UChVector
&input)
{
	layerOutput[0] = input;
	activatedLayerOutput[0] = input;

	for (int i=1; i<nLayers; i++)
	{
		layerOutput[i] = weights[i-1]*activatedLayerOutput[i-1];
		ActivateLayer(i);

	}

	return activatedLayerOutput[nLayers-1];
}

void UChFeedForwardNeuralNetwork::ActivateLayer(int layerIdx)
{
	int len = layerOutput[layerIdx].getLen();

	for (int i=0; i<len; i++)
	{
		switch(activationFunctions[layerIdx])
		{
			case LINEAR:
				activatedLayerOutput[layerIdx][i] =
layerOutput[layerIdx][i];
				break;

			case SIGM:
				activatedLayerOutput[layerIdx][i] = 1/(1+exp(-
layerOutput[layerIdx][i]));
				break;

			case TANH:
				activatedLayerOutput[layerIdx][i] = (1-exp(-
2*layerOutput[layerIdx][i]))/(1+exp(-2*layerOutput[layerIdx][i]));
				break;
		}

	}

}

void
UChFeedForwardNeuralNetwork::SetActivationFunction(ActivationFunction
newFunction)
{
	for (int i=0; i<nLayers; i++)
	{
		activationFunctions[i] = newFunction;

	}
```

```
}

void
UChFeedForwardNeuralNetwork::SetActivationFunction(ActivationFunction
newFunction, int numLayer)
{
	activationFunctions[numLayer] = newFunction;
}
```

MULTI RESOLUCIÓN

```
class UChSegmentationZone {

public:
	UChSegmentationZone();
	~UChSegmentationZone();

	UChDynamicArray<UChRunSegment> runCodeSegmentedImage;
	int initPoint;
	int endPoint;

	int resolution;

};

UChColorPixel UImageReference::GetImagePixel(int row, int col)
{
	UChColorPixel imagePixel;

	imagePixel.p1 = imageData[((row)*width+(col))*3];
	imagePixel.p2 = imageData[((row)*width+(col))*3+1];
	imagePixel.p3 = imageData[((row)*width+(col))*3+2];

	return imagePixel;
}

void UChImageColorSegmentation::ImageColorSegmentation(UChCameraState
cameraState)
{

	cameraState.horRight = cameraState.horRight;
	cameraState.horLeft = cameraState.horRight;

	int dZone = 30; //tamaño en píxeles de una zona
	int horMed = (cameraState.horLeft+cameraState.horRight)/2;

	int zonaHor = horMed/dZone;   // Zona en la que se encuentra el
horizonte

	// Calcula el numero de zonas
	int nZonas;
	if (horMed != cameraState.imageWidth/2-1)
		nZonas = 8;
	else
		nZonas = 7;
```

```
	UChDynamicArray<UChSegmentationZone> zones;
	zones.SetSize(nZonas);

	// Resolucion y limites zona 0
	zones[0].resolution = (zonaHor<5)?pow(2.0,(double)zonaHor):16;
	zones[0].initPoint = 0;

	if (zonaHor==0)
		zones[0].endPoint = horMed + dZone - horMed%2;
	else
		zones[0].endPoint = horMed - zonaHor*dZone - (horMed -
zonaHor*dZone)%zones[0].resolution;

	// Resolucion y limites zonas
	for(int i=1;i<nZonas;i++)
	{
		// Se calcula la resolucion de cada area
		zones[i].resolution = (abs(i-
zonaHor)<5)?pow(2.0,(double)abs(i-zonaHor)):16;

		// Calcula los puntos lémites de cada zona
		if(i==zonaHor)
		{
			zones[i].initPoint = zones[i-1].endPoint;
			zones[i].endPoint = zones[i].initPoint + 2*dZone;

		}
		else
		{
			zones[i].initPoint = zones[i-1].endPoint;
			zones[i].endPoint = zones[i].initPoint + dZone -
(zones[i].initPoint + dZone)%zones[i].resolution;

		}
	}
	// Ajuste para ultima zona
	if (zones[nZonas-1].endPoint<cameraState.imageHeight)
		zones[nZonas-1].endPoint = zones[nZonas-
1].endPoint+(cameraState.imageHeight-zones[nZonas-
1].endPoint)+(cameraState.imageHeight-zones[nZonas-
1].endPoint)%zones[nZonas-1].resolution;

	// Segmentacion y Runlength
	UChColorPixel pixelStructure;
	int pixelC1;
	int pixelC2;
	int pixelC3;
	//UChColorSegmentationLUT colorCode;

	imageReference->GoInitImagePixel();
	unsigned char *ptr =
colorSegmentedImage.segmentedImageNoFiltered.imageData;
	unsigned char ch;

	UChRunSegment newSegment;
```

```
	unsigned char imij, imij1;
	int cont = 1;
	unsigned char amarillo = YELLOW_CLASS;
	unsigned char cyan = CYAN_CLASS;
	unsigned char naranjo = ORANGE_CLASS;

	for (int k=0;k<nZonas;k++)
	{
		zones[k].runCodeSegmentedImage.SetSize(0);

		for (int i=zones[k].initPoint ; i<zones[k].endPoint;
i+=zones[k].resolution)
		{
			for (int
j=0;j<cameraState.imageWidth;j+=zones[k].resolution)
			{
				pixelStructure = imageReference-
>GetImagePixel(i,j);
				pixelC1 = (pixelStructure.p1/4 );
				pixelC2 = (pixelStructure.p2/4 );
				pixelC3 = (pixelStructure.p3/4 );

				imij =
colorCode.colorClass[pixelC1][pixelC2][pixelC3];

				pixelStructure = imageReference-
>GetImagePixel(i,j+zones[k].resolution);
				pixelC1 = (pixelStructure.p1/4 );
				pixelC2 = (pixelStructure.p2/4 );
				pixelC3 = (pixelStructure.p3/4 );

				imij1 =
colorCode.colorClass[pixelC1][pixelC2][pixelC3];


				if (IS_BLOB_CLASS(imij))
				{
					if (imij==imij1)
					{
						cont++;
					}
					else
					{
						newSegment.x  = j - cont + 1;
						newSegment.y  = i;
						newSegment.color = imij;
						newSegment.length = cont;
						newSegment.label = 0;

	zones[k].runCodeSegmentedImage.push(newSegment);
						cont = 1;
					}
				}
			}
		}
	}
```

```
}
```