VNIVERSIDAD NACIONAL AVTÓNOMA DE MEXICO

UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

POSGRADO EN CIENCIA E INGENIERÍA DE LA COMPUTACIÓN

"ESTIMACIÓN DE LA POSICIÓN DEL ROBOT CON PROCESAMIENTO DIGITAL DE IMÁGENES Y TÉCNICAS DE RECONOCIMIENTO DE PATRONES USANDO CUANTIFICACIÓN VECTORIAL Y REDES NEURONALES"

T E S I S

QUE PARA OBTENER EL GRADO DE:

MAESTRO EN INGENIERÍA (COMPUTACIÓN)

P R E S E N T A:

VÍCTOR JAIR GUTIÉRREZ GONZÁLEZ

DIRECTOR DE TESIS: JESÚS SAVAGE CARMONA

México, D.F. **2007.**

AGRADECIMIENTOS

A mi Abuelita, a mi Hermana y a mi Papá por su apoyo.

A Karen, Silvia, César y Alejandro por su amistad.

A mi tutor el Dr. Jesús Savage Carmona por su paciencia y confianza.

A mis maestros del Posgrado y en especial a mis sinodales el Dr. Ángel Fernando Kuri Morales, el Dr. Boris Escalante Ramírez, el Dr. Carlos Rivera Rivera y el Dr. Ernesto Bribiesca Correa por su tiempo, por su orientación y por compartir su conocimiento.

A cada uno de los trabajadores del IIMAS por su atención y a mis compañeros del Laboratorio de Bio-Robótica de la Facultad de Ingeniería.

En especial a Dios y a mi Mamá por estar conmigo en todo tiempo y por sus consejos:

"Todo lo que te viniere a la mano para hacer, hazlo según tus fuerzas; porque en el Seol, adonde vas, no hay obra, ni trabajo, ni ciencia, ni sabiduría." Ec. 9:10

"No temas, porque yo estoy contigo; no desmayes, porque yo soy tu Dios que te esfuerzo; siempre te ayudaré, siempre te sustentaré con la diestra de mi justicia." Is. 41:10

"Porque Jehová da la sabiduría, y de su boca viene el conocimiento y la inteligencia. El provee de sana sabiduría a los rectos; es escudo a los que caminan rectamente." Pr. 2:6-7

"Enséñanos de tal modo a contar nuestros días, que traigamos al corazón sabiduría." Sal. 90:12

Sólo a Dios sea la gloria, la honra, la alabanza y la adoración

Muchas Gracias Dios

ÍNDICE GENERAL

INTRODUCCIÓN

En el laboratorio de Bio-Robótica, de la Facultad de Ingeniería, se tiene un equipo de robots de la categoría de fútbol "small size" para la competencia del ROBOCUP, en la cual se utilizan 2 cámaras para detectar a los robots y dar su posición.

Actualmente, esto se realiza asociando cada cámara a un sistema de visión, y cada sistema de visión a una sola computadora. La detección de los robots se efectúa a través de parches de colores que éstos traen en la parte superior (ya que las cámaras se colocan en el techo) figura (1). Estos parches también ayudan a identificar el tipo de robot (portero o jugador libre), a qué equipo pertenece, en qué parte de la cancha se encuentra y su ángulo de orientación.

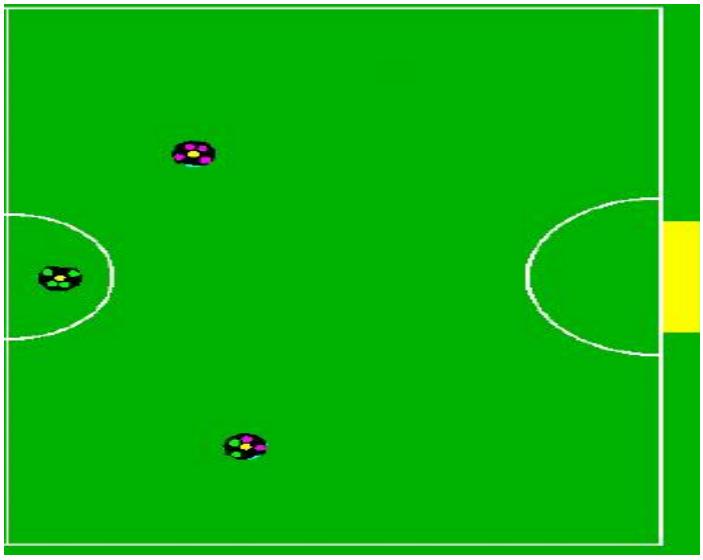


Figura 1. Vista superior de los robots

Esta competencia va evolucionando año con año, por lo cual, hoy en día se tienen que emplear cámaras que utilizan lente gran angular ya que las dimensiones de la cancha han crecido. El uso de estos lentes produce una distorsión a la imagen y por lo tanto distorsiona la posición de los robots. Si no se usaran estas cámaras con lentes gran angular, entonces se tendrían que utilizar, por lo menos, el doble de cámaras sin lente gran angular para poder cubrir la cancha de fútbol.

A continuación se ilustran los pasos fundamentales del Procesamiento Digital de Imágenes.

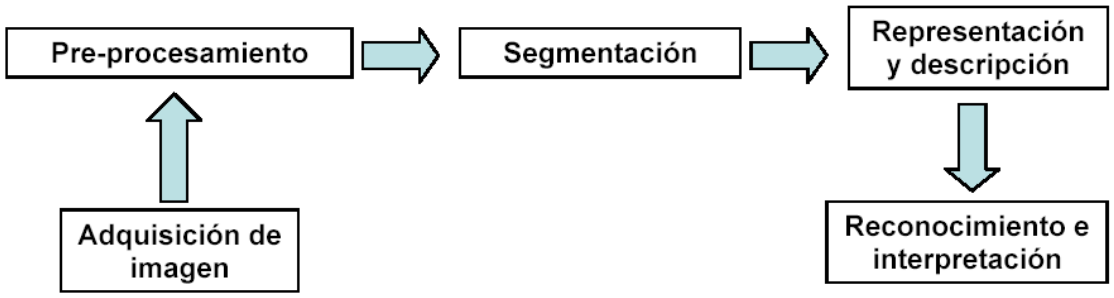


Figura 2. Diagrama de los pasos fundamentales del Procesamiento Digital de Imágenes[2]

• El primer paso es la adquisición de la imagen; para lograrlo, se requiere de una cámara y la capacidad de digitalizar la señal producida por ella. En este paso es donde se incrementará el número de cámaras cuando sea necesario.

• Después de que una imagen digital ha sido obtenida debe ser procesada, con la finalidad de mejorarla y, de esta manera aumentar la probabilidad de éxito en las etapas restantes. Algunos ejemplos de métodos utilizados en esta etapa son, entre otros, aumentar el contraste y remover el ruido. En esta etapa se eliminará la distorsión producida por el lente gran angular.

• La siguiente etapa es la segmentación en donde se realiza una partición de la imagen en sus partes constituyentes u objetos. En esta etapa es donde se utilizará la clasificación hecha por la cuantificación vectorial [4] o por la red neuronal [8].

• Los datos obtenidos en la etapa de segmentación deben ser convertidos a una estructura apropiada para su posterior procesamiento. La descripción, también llamada selección de características, extrae información cuantitativa o características relevantes para diferenciar un objeto de los demás.

• La última etapa es el reconocimiento e interpretación. El reconocimiento es el proceso que asigna un identificador a un objeto basado en la información provista por sus descriptores. La interpretación es asignar un significado al objeto reconocido. En general, los sistemas de procesamiento que incluyen reconocimiento e interpretación están asociados a aplicaciones de análisis de imágenes cuyo objetivo es la extracción automática de información.

Objetivos

1. Corregir la distorsión producida por el lente gran angular

Dos tipos de distorsión aparecen en una gran cantidad de lentes, especialmente en lentes del tipo gran angular, que son la distorsión de Barril (Barrel) [14] y la distorsión de Cojín (Pincushion) [14]. Este error se corregirá moviendo cada píxel de una manera radial, con base en una función polinomial de cuarto grado cuyos coeficientes dependen de cada lente. Esta función relaciona la distancia de un píxel del centro de la imagen origen r_{src} a la imagen destino r_{dest} [12]:

$$r_{src} = (a* r^3_{dest} + b*r^2_{dest} + c*r_{dest} +d)*r_{dest}$$

El uso de esta función va acompañado del procesamiento de imágenes para analizar patrones.

Esta forma de quitar la distorsión del lente gran angular es más general que el método de Tsai [2] ya que no es necesario conocer todas las características de la cámara ni las distancias exactas que existen de la cámara al objeto a reconocer. Ya que solo colocando un patrón como un tablero de ajedrez y a través del procesamiento de la imagen se podrá eliminar la distorsión del lente gran angular.

Este procesamiento de la imagen dará con mayor exactitud la posición del robot y no debe ser computacionalmente costoso para la computadora.

2. Mejorar el muestreo y la clasificación de colores.

Las técnicas de reconocimiento de patrones requieren de muchas muestras para poder lograr un mejor entrenamiento. Estas muestras se obtienen al colocar patrones de colores y mediante un procesamiento sencillo de la imagen poder tomar la mayor cantidad de muestras de cada color a utilizar.

- Cuantificación Vectorial: Se usará para mejorar el reconocimiento de colores.

Ésta utilizará:

- El Método de Splitting [4], el cual permite crear codebooks [4] de potencia de 2, ya que se trata de un proceso iterativo en el que en cada iteración se dobla el tamaño del codebook obtenido en la iteración anterior. La mejora del codebook es a través del Algoritmo de Lloyd [4].

- Red Neuronal: Se usará para el reconocimiento de colores de los parches.

Ésta utilizará:

- La Red Neuronal del tipo feedforward. Esta red tiene ligas unidireccionales, sin ciclos. Normalmente estas redes están arregladas en capas. Cada unidad está ligada solo con las unidades de la siguiente capa. No hay ligas inter-capas, ni ligas a capas anteriores, ni ligas saltándose capas. Una red feedforward calcula una función de las entradas que depende de los pesos.

Al ejecutar estas 2 técnicas para el reconocimiento de colores, se comparará la eficiencia en el reconocimiento y la ejecución en tiempo real para la competencia del ROBOCUP.

Estas técnicas permitirán una mejor clasificación ya que en la competencia las características de los colores varían dependiendo de la intensidad de la luz (ya que la iluminación no es uniforme en la cancha) y estas técnicas se pueden adaptar fácilmente a estos cambios.

3. Utilizar más de un dispositivo de captura en el sistema de visión.

En este paso se necesita saber utilizar las clases proporcionadas por Directx9 [5] para la captura de video y para el acceso a los dispositivos de captura de video del sistema operativo (Windows).

Al saber cómo el sistema operativo enumera los dispositivos, entonces se sabrá cuántos dispositivos de video están siendo utilizados en una sola computadora y así poder mandar llamar los datos capturados por la(s) cámara(s) al sistema de visión. Con esto, las variables del programa del sistema de visión pasan a ser arreglos para manejar más de una cámara.

Por lo que el sistema podrá trabajar por lo menos con 2 cámaras en un solo sistema de visión y por lo tanto en una sola computadora.

Con lo mencionado anteriormente se mejorará el sistema de visión del equipo de fútbol de small size del ROBOCUP.

Trabajos anteriores

En [1] se tiene un trabajo de visión para la ROBOCUP en la Small League desarrollado en el laboratorio de Bio-Robótica, este sistema está basado en [2]. En ambos trabajos se describe el sistema de visión. El sistema de visión desarrollado [1] hace uso de webcams para el puerto USB, lo que permitiría que no sólo instituciones con

recursos limitados, sino también entusiastas de la robótica puedan iniciar sus propios proyectos.

Otros sistemas de visión han sido implementados para ROBOCUP por algunas universidades de distintos países del mundo, sin embargo, estos sistemas se basan en dispositivos de hardware especializados [20] que son difíciles de conseguir y tienen alto costo. Las cámaras digitales de alta resolución, les permite salvar algunos obstáculos, como imágenes ruidosas, al momento de hacer su procesamiento. Aunque este planteamiento es el más práctico, no siempre se cuenta con tal equipo, ya que también se requieren tarjetas de captura de video digital. Además, no son software libre ni están disponibles al público, o si lo están, son librerías de bajo nivel para procesamiento digital de imágenes en tiempo real [19] que requieren de soluciones adicionales para su implementación dentro del proyecto RoboCup.

Para la elección de un espacio de color apropiado, se balancea lo que el hardware provee con lo que sería favorable para la representación del umbral. Primero, se considera [18] el espacio de color RGB, el cual es un formato común para desplegar y manipular una imagen, y es provisto directamente por la mayoría del hardware de video de captura. Su problema principal se ubica en el valor de intensidad de la luz y sombra siendo esparcidas a través de los tres parámetros. Esto hace difícil separar la varianza en la intensidad desde la varianza del color con un umbral rectangular. El espacio de color RGB es usado con una proyección que llaman fraccional YRGB donde Y=(R+G+B)/3 R=R/Y, G=G/Y, B=B/Y, este espacio probó [18] ser robusto para describir los colores con umbrales de cubo donde RGB es el único espacio de color disponible.

El espacio de color YUV es usado por los estándares de video NTSC y PAL. Este espacio consiste de un valor de intensidad (Y), y dos valores de cromaticidad (U,V). Este es usado en estándares de video debido a que su clasificación es cercana a la percepción humana del color, y éste es provisto directamente por la mayoría de los dispositivos de video análogos. Desde que la intensidad es separada en su propio parámetro, la causa principal de correlación entre los valores de color de la componente de intensidad ha sido removida, y ésta es la mejor representación para umbrales rectangulares. YUV probó [18] ser robusto en general y rápido ya que este es provisto directamente por hardware.

Una solución para el cambio de intensidad en el campo de juego consiste en partir el área del campo en cuadros suficientemente pequeños para considerar la iluminación uniforme en su superficie, medir el valor de las marcas de color cuando está situado en esa región, y de esta manera construir un mapa de color, el cual puede ser usado después como referencia en el proceso de reconocimiento de blobs [17].

Otro método para clasificar los colores requiere modelos geométricos de los objetos los cuales son colores para ser calibrados. En la implementación [21], un modelo

de líneas de campo en las coordenadas del mundo, y una de las bolas en las coordenadas de la imagen fue usada. Esta calibración se hizo de 4 colores simbólicos distintos posibles: campo (verde), bola (naranja), amarillo meta (amarillo) y azul meta (azul). El método efectúa esto haciendo cualquier suposición específica acerca del color del campo, bola o metas excepto que ellas son de homogeneidad áspera en colores claros, y que las líneas del campo son relativamente brillosas al campo. El trabajo de calibración para la localización de robots (sin usar información de color), es entonces creciendo las regiones de color homogéneas y marcando su tamaño y forma. Si una región marca uno de los esperados, su color es añadido a la clase de color simbólica respectiva. Este método puede ser corrido en un hilo de fondo para habilitar al robot rápidamente para recalibrar en respuesta a los cambios de iluminación. El método fue probado en el campeonato Mundial de ROBOCUP en Osaka 2005 por el equipo de Mid-Size de los FU-Fighters en el cual obtuvo el segundo lugar.

La mayoría del trabajo hecho en la calibración del color en el ROBOCUP ha sido enfocado en asignar cada punto en el espacio de color específico a un color simbólico especifico. Esto requiere una tabla la cual permite una búsqueda de color simbólico (verde, naranja, amarillo, etc.) de cualquier valor de color en un espacio de color en particular (RGB; YUV; etc.). Sin embargo, mucho trabajo se enfoca en el llenado automático de estas tablas de búsqueda.

Mayer [24], presenta un método fuera de línea, semi-autónomo en la calibración del color, implementado en la liga de Mid-Size. Retinex [21] es usado para mejorar la constancia del color, y el agrupamiento de k-means es usado para la adaptación de las clases de color. Este método analiza la vecindad de los colores en el espacio de color y entonces forma grupos que representan clases de color. En contraste, el método presentado en [21] confía en la forma geométrica esperada del objeto perteneciente a la clase de color y no confía en el análisis del espacio de color. Además, esto es totalmente automático, no requiere mapeo manual para los colores simbólicos.

Un método llamado KADC(Conocimiento Basado en la Calibración de Color Dinámica Autónoma) es presentado en [25]. KADC es un método para la clasificación de color autónomo en línea, implementado en la liga Sony Legged. KADC también utiliza la información geométrica del campo para definir las clases de color, y entonces actualiza con la ayuda de un agrupador de color con una métrica similar llamada EMD. El método presentado en [21] está también basado en el conocimiento geométrico del campo, pero éste es combinado con la detección menor de color de la posición del robot. Entonces la clasificación es actualizada usando sólo criterios geométricos sin tener que incorporar ninguna métrica similar a la clasificación establecida previamente. Esto permite la dirección de un abrupto incremento / decremento en la iluminación, la cual se reporta [21] como un problema cuando se aplica KADC.

Algunos sistemas de visión en ROBOCUP utilizan diferencias relativas entre áreas de color. En estos sistemas la calibración es integrada en el paso de la detección de objetos o no se requiere la calibración.

Otra área de importante investigación son los algoritmos de constancia de color. Si se da una escena bajo iluminación estándar, la misma escena experimentará un cambio de valor de colores en el espacio de color cuando se sujeta a una diferente iluminación desconocida. El propósito de los métodos de la constancia de color en la visión por computadora es resolver este problema convirtiendo cada valor del espacio de color (ej. RGB) a los que sería bajo el estándar de iluminación. Estos métodos de la constancia de color pueden hacer los sistemas de visión en el ROBOCUP robustos ante los cambios de iluminación, pero la calibración de color aún tiene que ser hecha para la iluminación estándar (manual o automática). Desde que el color del objeto en ROBOCUP varía desde un lado de la competición a otro, la calibración manual es aún muy inconveniente. Por lo tanto, ha habido más interés en los métodos de auto calibración en el ROBOCUP.

Austermeier [26] encontró la correspondencia de color entre 2 ajustes de la iluminación usando mapas de características auto organizados (SOM) en el espacio de color. Dos SOMs son construidos, uno para la nube de puntos de color bajo la referencia de iluminación inicial y otra para los nuevos ajustes de iluminación. La distancia entre la localización correspondiente en la rejilla en los 2 mapas es entonces usada como un vector de corrección para el conjunto de puntos de color pertenecientes a ese elemento del volumen. Desafortunadamente, el método es computacionalmente muy caro.

Otra clase de métodos de constancia de color, están basados en la suposición del mundo gris de G. Buchsbaum [27] y referidos como métodos de un mundo gris. Ellos están basados en la suposición de que el promedio de la reflexión de la superficie de una escena típica es gris (o alguna otra constante de color). Este método tiene efectos no deseados en un escenario de ROBOCUP, una alternativa para los métodos del mundo gris es colocar parches blancos en el robot, la imagen es entonces corregida globalmente hasta que el área de la imagen es ocupada por el parche que ha alcanzado su valor objetivo. Este método no es afectado por un objeto oscuro largo o brilloso en la imagen. Sin embargo, la iluminación no uniforme tal como una sombra o una falla en el brillo de la luz en el parche, predispondrá la imagen globalmente. Una implementación de este método fue usado en el equipo Mid-Size FU-Fighters, donde una pieza de papel blanco fue unida alrededor del lente de la cámara. El balance del blanco fue alcanzado ajustando los parámetros de la cámara con la ayuda de 2 controladores PID [28].

Por último cabe mencionar que la eliminación de la distorsión en su mayoría es a través del método de Tsai [2] y del método que Von Seidel [22] expresó mediante el desplazamiento de un punto en una imagen ideal como una serie de Taylor en su distancia radial, desde el centro de la distorsión.

CAPÍTULO 1

Distorsión

La distorsión tiende a ser más seria en ángulos extremadamente anchos, telefoto, y lentes de zoom. Ésta es muy inconveniente en la fotografía arquitectural y en la fotografía usada para mediciones (meteorología). Ésta puede ser altamente visible en líneas tangenciales cerca del límite de la imagen, pero no es visible en líneas radiales. En una distorsión de lentes bien centradas es simétrica alrededor del centro de la imagen. Pero estos lentes pueden ser descentrados debido a la pobre calidad de manufactura o un daño por golpe.

En el modelo más simple de distorsión de la lente, el radio no distorsionado y el radio distorsionado r_{nd} y r_d (distancias desde el centro de la imagen) son relacionados por la ecuación [14]:

$$r_{nd} = r_d + k_1 r_d^3 \tag{1.1}$$

donde $k_1 > 0$ para la distorsión de barril y $k_1 < 0$ para la distorsión de cojín.

Esta ecuación (1.1) de tercer orden es una de las Aberraciones de Seidel, las cuales son aproximaciones polinomiales de orden-bajo para las degradaciones de las lentes. Otras aberraciones incluyen astigmatismo, coma, curvatura del campo, etc. La aproximación de tercer orden es suficiente para muchas lentes.

Para medir la distorsión, se necesitará un patrón de rejilla rectangular o cuadrado.

1.1 Las Cinco Aberraciones de Seidel

Los 5 tipos básicos de aberración que son originados por la geometría de las lentes o espejos, y que son aplicables a sistemas que se comportan con luz monocromática, son conocidos como las aberraciones de Seidel. En 1857, fueron descritas, en un artículo, por Ludwing Von Seidel. Estas son las aberraciones que llegan a ser evidentes en ópticas de tercer orden, también conocidas como ópticas de Seidel.

A continuación se da una definición numérica de las funciones seno y coseno que es independiente de los conceptos geométricos. Esta definición se da por medio de series finitas de potencias. Por lo que las siguientes series de potencias convergen a *sen(x)* y *cos(x)* respectivamente.

$$sen(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \frac{x^9}{9!} - \frac{x^{11}}{11!} + \frac{x^{13}}{13!} - \cdots$$

y

$$\cos(x) = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \frac{x^8}{8!} - \frac{x^{10}}{10!} + \frac{x^{12}}{12!} - \cdots$$

Con esta definición numérica se dará una breve explicación de los órdenes de la ópticas de las lentes.

Cuando se desatienden los términos lejanos en la serie, se procede como si *sen(x)=x*, y *cos(x)=1* y se obtienen ópticas de primer orden, en las cuales todas las lentes son perfectas. Cuando se incluyen los términos cuadrado de *x* y cúbico de *x*, entonces se ha procedido a ópticas de tercer orden, en la cual llegan a ser evidentes las aberraciones que resultan de la naturaleza de las lentes reales, exclusivas de aberraciones cromáticas.

Las cinco aberraciones de Seidel son:

- Aberración esférica: Esta es la aberración que afecta a los rayos desde un punto en el eje óptico; los rayos, desde este punto, van hacia afuera en diferentes direcciones pasando a través de diferentes partes de la lente, entonces, si la lente es esférica, o no tiene la forma necesaria para traer a todos ellos para un enfoque, entonces no todos estos rayos estarán enfocados al mismo punto en el otro lado de la lente.

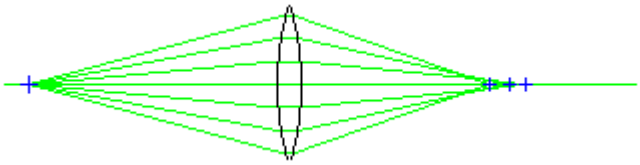


Figura 1.1: Aberración Esférica [29]

- Coma: Esta aberración afecta los rayos desde puntos fuera del eje óptico. Si la aberración esférica es eliminada, diferentes partes de la lente lleva los rayos desde el eje a el mismo enfoque. Pero el lugar donde la imagen de un punto fuera del eje es formada podría continuar cambiando cuando las diferentes partes de las lentes están consideradas.

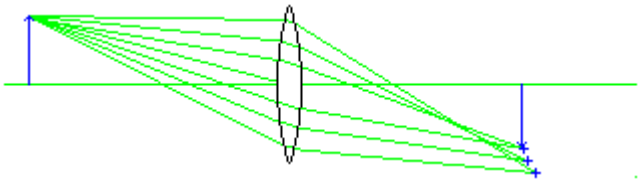


Figura 1.2: Aberración de Coma [29]

- Astigmatismo: Esta es otra aberración que afecta los rayos desde un punto fuera del eje óptico. Estos rayos, que se mandan a través de la lente a el punto en la imagen donde ellos estarán enfocados, pasan a través de una lente que es, desde su perspectiva, inclinado. Aunque ni la aberración esférica ni el coma previenen esto de llegar a un enfoque angulado, si se consideran los rayos de luz que están en el plano de la inclinación, y los rayos de la luz que están en el plano perpendicular a esto, entonces estos rayos pasan a través de una parte del lente con un perfil diferente. Por lo que ellos no podrían estar enfocados en la misma distancia desde la lente, aun cuando ellos no lleguen a un enfoque en cada caso.

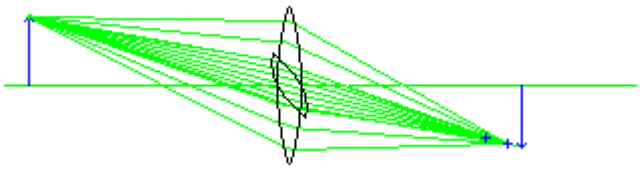


Figura 1.3: Aberración de Astigmatismo [29]

- Curvatura del campo: Aún cuando la luz desde cada punto en el objeto es llevar a un enfoque angulado, los puntos en los cuales ellos llevan el enfoque podrían mentir sobre una superficie curveada en lugar de un plano extendido.

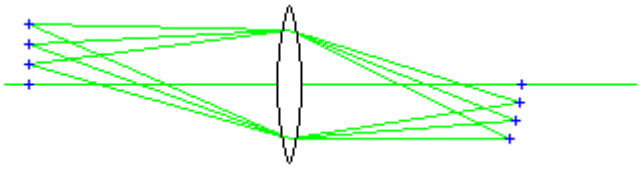


Figura 1.4: Aberración de Curvatura del Campo [29]

- Distorsión: Aún cuando todas las aberraciones previas han sido corregidas, la luz desde los puntos en el objeto podrían ser llevados juntos en el plano de la imagen a la distancia errónea desde el eje óptico, en lugar de ser linealmente proporcional a la distancia desde el eje óptico en el objeto. Si la distancia

incrementa más rápido que en el objeto, se tiene la distorsión de cojín, si la distancia va más despacio, se tiene la distorsión de barril.

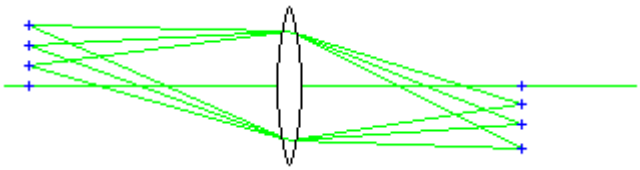


Figura 1.5: Aberración de Distorsión [29]

1.2 Corrección de la distorsión producida por la lente gran angular.

Dos tipos de distorsión aparecen en una gran cantidad de lentes, especialmente en lentes del tipo gran angular, que son la distorsión de Barril (Barrel) y la distorsión de Cojín (Pincushion).

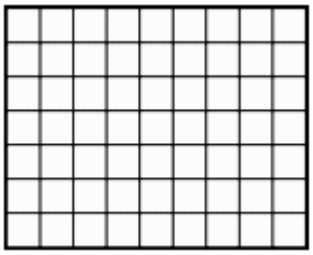


Figura 1.6: Imagen sin distorsión

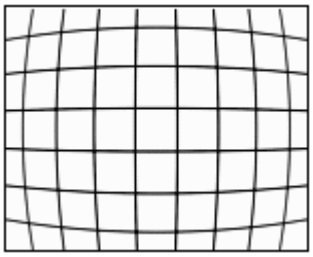


Figura 1.7: Imagen con distorsión de barril

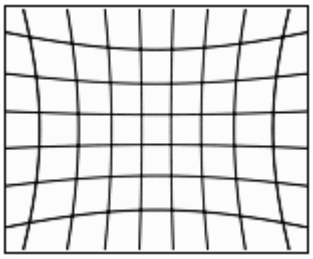


Figura 1.8: Imagen con distorsión de cojín

Este error se corrige moviendo cada píxel de una manera radial, con base en una función polinomial de tercer orden cuyos coeficientes dependen de cada lente. Esta

función relaciona la distancia de un píxel del centro de la imagen origen r_{src} a la distancia correspondiente en la imagen destino r_{dest} :

$$r_{src} = \left(a * r^3{}_{dest} + b * r^2{}_{dest} + c * r_{dest} + d\right) * r_{dest} \qquad (1.2)$$

Al utilizar la fórmula (1.2) los radios r_{src} y r_{dest} están normalizados de *[0,1]*. Ahora, si en la fórmula (1.2) se tiene $r_{dest} = 1$, que es el límite de la imagen, se observa que, para no escalar la imagen, debe cumplirse que *a+b+c+d=1*, por lo que *d=1-(a+b+c)*.

Si se utiliza *d=1*, y *a=b=c=0* se tiene $r_{src} = r_{dest}$ por lo cual el parámetro "d" es llamado el "factor de escalamiento" de la imagen. De aquí, que al escoger otro valor de "*d*" la imagen se escala y cualquier otro valor de los parámetros *a*, *b* y *c* distorsionan la imagen, por lo cual se conocen como "parámetros de distorsión".

Dada la relación *d=1-(a+b+c)*, se tiene que, al hacer a los parámetros de distorsión negativos *(-a, -b y -c)*, se desplazan los puntos distantes hacia el centro corrigiendo así la distorsión de cojín. Ahora, haciendo los parámetros de distorsión positivos *(+a, +b y +c)*, se desplazan los puntos distantes lejos del centro de la imagen, corrigiendo así la distorsión de barril.

Por lo mencionado anteriormente, se tiene que los parámetros *a, b y c,* de la ecuación (1.2) se comportan como el parámetro k_1 de la ecuación (1.1). Por lo que la ecuación (1.1) es una de las opciones que tiene la ecuación (1.2).

Corrigiendo con el parámetro "*a*" se afecta solamente a los píxeles mas lejanos de la imagen, mientras que la corrección con "*b*" es más uniforme. Finalmente, se puede corregir la distorsión de "cojín" y de "barril" en la misma imagen: Si las regiones más alejadas exhiben la distorsión de "cojín" y las regiones interiores exhiben la distorsión de "barril", se deben usar los valores de "a" negativo y "b" positivo .

En la mayoría de los casos, se consiguen resultados satisfactorios usando solo un parámetro.

A continuación se muestra un ejemplo para explicar las variables de distorsión:

Primero definimos una función la cual toma las funciones del radio.

$$y = f(x), \ \ con \ \ f(x) = x, f(x) = x^2, f(x) = x^3, f(x) = x^4$$

	x=0.1	x=0.2	x=0.3	x=0.4	x=0.5	x=0.6	x=0.7	X=0.8	x=0.9
x	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
x^2	0.01	0.04	0.09	0.16	0.25	0.36	0.49	0.64	0.81
x^3	0.001	0.008	0.027	0.064	0.125	0.216	0.343	0.512	0.729
x^4	0.0001	0.0016	0.0081	0.0256	0.0625	0.1296	0.2401	0.4096	0.6561

Tabla 1.1: Muestra los valores del radio normalizado de una circunferencia unitaria

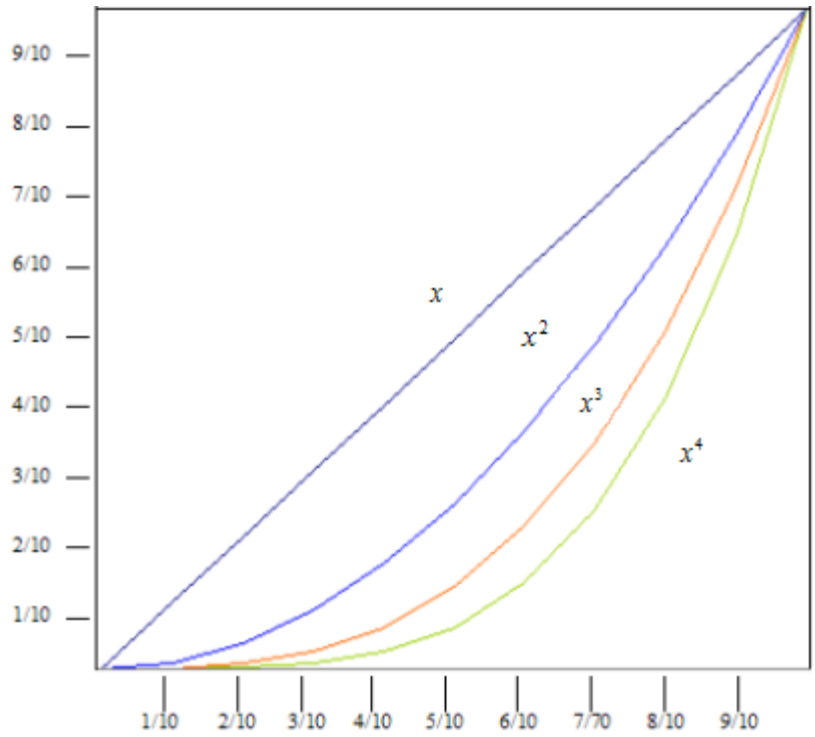


Figura 1.9: Gráfica de la Tabla 1.1

Las funciones graficadas en la figura (1.9) tienen un rango de [0,1]. La línea de la función cuadrática muestra que mientras el radio esté entre [0,5/10] casi no afecta a la transformación del radio, por eso se dice que el parámetro "a" afecta a los píxeles más lejanos. La línea de la función cúbica, indica que mientras el radio está entre [0,4/10] casi no afecta a la transformación del radio. La línea de la función cuadrática, muestra que la transformación del radio es creciente de manera constante a diferencia de la función cúbica y cuadrática. La línea de la función lineal hace referencia al factor de escalamiento.

1.3 Coordenadas Polares

El sistema cartesiano no es la única manera de fijar la posición de un punto en un plano, otro método o sistema usado es el de las coordenadas polares.

En el sistema de coordenadas polares los elementos de referencia son un punto llamado polo (O) y una semirrecta de origen al polo llamada eje polar (e).

Las coordenadas de un punto son:

Distancia del polo al punto. Se llama radio vector del punto y se suele representar por la letra r o la letra griega ρ.

El ángulo que forma el radio vector con el eje polar se llama argumento o ángulo polar y se le suele representar por la letra θ u ω.

El argumento se mide a partir del eje polar positivamente en sentido contrario al de las manecillas del reloj (sentido directo) y negativamente en el mismo sentido de las manecillas del reloj (sentido retrógrado).

El radio vector se considera positivo si se mide donde termina el argumento y negativo si se mide en su prolongación.

El argumento se puede dar en medidas angulares o circulares (radianes).

En el sistema cartesiano se establece una correspondencia biunívoca entre los puntos del plano y pares de números reales dados en un orden. Pero a cada par de coordenadas polares le corresponde un solo punto, en cambio a cada punto le corresponden diversos pares de coordenadas polares.

1.3.1 Localización de puntos

Para localizar un punto, conocidas sus coordenadas, se traza un ángulo igual al argumento y después se lleva, a partir del polo y en el sentido que indique el signo, sobre el lado terminal o su opuesto, según sea positivo o negativo, la longitud del radio vector.

1.3.2 Relación entre las coordenadas cartesianas y las polares

Supongamos dos sistemas de coordenadas, uno cartesiano y otro polar, con el mismo origen y el eje polar coincidiendo con la parte positiva del eje X. Si (x, y) y (r, θ) son las coordenadas de un mismo punto P del plano en los dos sistemas, se tiene

$$x = r\cos\theta \ , \qquad y = r\,sen\ \theta ,$$

fórmulas que dan las coordenadas cartesianas de un punto cualquiera en función de las polares.

Para obtener las polares en función de las cartesianas, se observa:

$$sen\ \theta = y/r \ , \ \ cos\ \theta = x/r$$

$$r^2 = x^2 + y^2 \qquad r = \sqrt{x^2 + y^2}$$

1.3.3 Gráfica de una ecuación en coordenadas polares

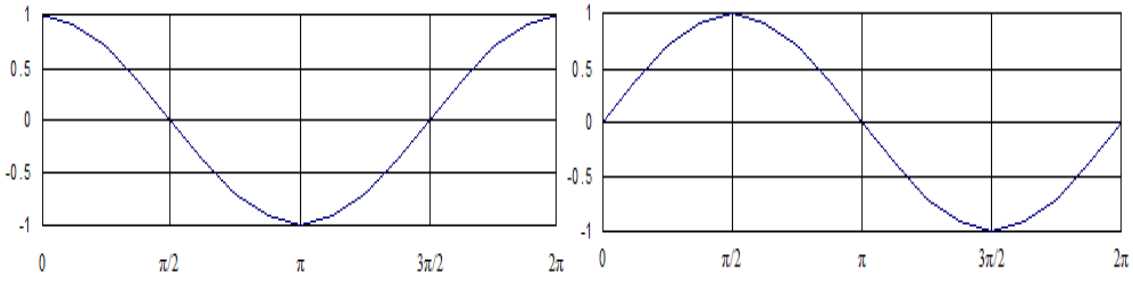
Igual que en las coordenadas cartesianas, la gráfica de una ecuación en coordenadas polares está formada por el conjunto de puntos cuyas coordenadas satisfacen la ecuación.

Para el trazado de la gráfica se construye una tabla de valores. Se dan valores al argumento θ y se calcula el correspondiente para el radio vector r.

1.4 Función seno y Función coseno

En vista de que para cualquier número real t se tiene $\cos t = x$ y $\text{sen}\, t = y$, las propiedades de las funciones circulares (seno y coseno), son:

1. cos *(-t)* = cos *t*	sen *(-t)* = - sen *t*
2. cos *(π - t)* = cos *t*	sen *(π - t)* = sen *t*
3. cos *(π + t)* = - cos *t*	sen *(π + t)* = - sen *t*
4. cos *[t + k(2π)]* = cos *t*	sen *[t + k(2π)]* = sen *t*



a) Función coseno *b)Función seno*

Figura 1.10: Gráficas de las funciones coseno y seno en el rango de 0 a 2π

1.5 Algoritmo para quitar la distorsión de una imagen

- Se obtiene la imagen en una representación de matriz.

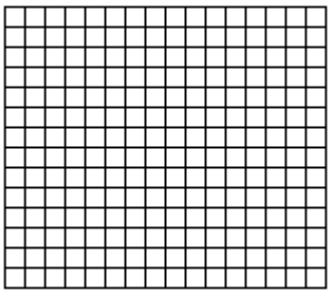


Figura 1.11: Matriz de una imagen

- Se obtienen las dimensiones de la imagen (matriz)

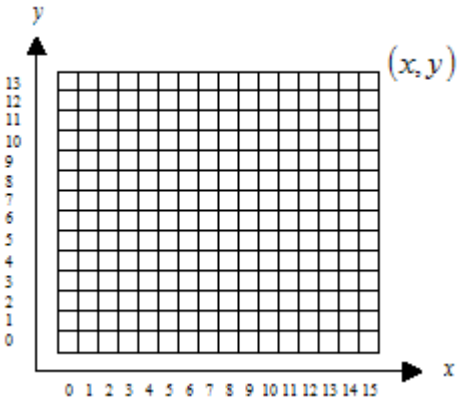


Figura 1.12: Matriz de una imagen con sus dimensiones

- Se obtiene el punto medio $\left(\frac{x}{2}, \frac{y}{2}\right)$ de las dimensiones de la imagen para ser el centro de la imagen, las coordenadas deben ser enteras por lo que se redondea hacia arriba

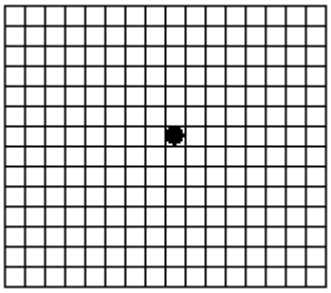


Figura 1.13: Matriz con su punto medio

- A partir del centro se obtienen todas las distancias (*radio(x,y)*) del centro a un punto de la imagen y también obtenemos la distancia mayor del centro a un punto de la imagen.

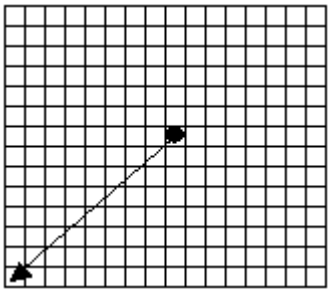
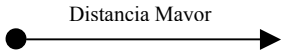


Figura 1.14: Matriz con distancia máxima desde el punto medio



- Con la distancia mayor normalizamos las distancias para que el radio vaya de [0,1]

$$radio\_normalizado(x,y) = \frac{radio(x,y)}{Dist.\ Mayor}$$

- Se obtiene el ángulo de cada punto a partir del centro representado en la imagen

$$\theta = a\cos\left(\left(x - centrox\right)/radio(x,y)\right)$$

- Se aplica la ecuación (1.2) a la distancia normalizada (*radio_ normalizado(x,y)*), siendo $r_{dest} = radio\_normalizado(x, y)$

$$r_{src} = \left(a * r^3{}_{dest} + b * r^2{}_{dest} + c * r_{dest} + d\right) * r_{dest}$$

- El resultado se desnormaliza

$$radio(x, y) = r_{src} * Dist.\ Mayor$$

- Cada punto resultante se transforma en coordenada cartesiana usando el ángulo

$$x = radio(x, y)\cos\theta + centrox \ , \qquad y = radio(x, y)sen\theta + centroy$$

CAPÍTULO 2

Técnicas de reconocimiento de patrones y clasificadores (Cuantificador Vectorial y RNA).

2.1 Fundamentos del color

El conocimiento adquirido sobre color, hace referencia al color pigmento que considera como colores primarios (aquellos que por mezcla producirán todos los demás colores) al rojo, el amarillo y el azul. En realidad existen dos sistemas de colores primarios: colores primarios luz y colores primarios pigmento.

El blanco y negro son llamados colores acromáticos, ya que los percibimos como "no colores".

Los colores producidos por luces (en el monitor de nuestra computadora, en el cine, en la televisión, etc) tienen como colores primarios, el rojo, el verde y el azul (RGB, por sus siglas en inglés, *Red Green Blue*) cuya fusión de éstos, crean y componen la luz blanca, por eso a esta mezcla se le denomina síntesis aditiva; y las mezclas parciales de estas luces dan origen a la mayoría de los colores del espectro visible.

Figura 2.1: Color luz, síntesis aditiva[11]

La caracterización de la luz es vital para la ciencia del color. Desde el punto de vista del ojo humano, todos los colores son vistos como combinaciones variables de los tres colores primarios: rojo (R), verde (G) y azul (B). Las características utilizadas generalmente para distinguir un color de otro son: brillo, matiz y saturación. El brillo incorpora la noción cromática de intensidad. El matiz es un atributo asociado con la longitud de onda dominante en la mezcla de longitudes de onda de la luz; viene a ser

como el color percibido por el observador; cuando se dice de un objeto que es rojo, naranja o amarillo, realmente se está especificando su matiz; el grado de saturación es inversamente proporcional a la cantidad de luz blanca añadida. El matiz y la saturación cuando se toman conjuntamente se denominan cromaticidad. Y, por tanto, un color se puede caracterizar por su brillo y su cromaticidad.

2.1.1 Modelos de color

El propósito de un modelo de color es facilitar la especificación de los colores de alguna forma estándar. En esencia, un modelo de color es una especificación de un sistema de coordenadas 3D y un subespacio dentro de dicho sistema donde cada color se representa por un punto [3].

Muchos modelos de color en uso, hoy en día, están orientados hacia el hardware, o hacia aplicaciones donde la manipulación del color es un objetivo. Los modelos más comunes orientados al hardware usados en la práctica son el RGB (rojo, verde, azul) para monitores en color y una amplia gama de videocámaras; el CMY (cyan, magenta, amarillo), para impresoras en color; y el YIQ (reflectancia o intensidad, infase y cuadratura), para la televisión a color. En procesamiento de imágenes se utilizan RGB, YIQ, HSV (matiz, saturación, valor) y HSI (matiz, saturación, intensidad). A continuación se introducen las características básicas de los modelos RGB y HSI.

2.1.1.1 El modelo RGB

En el modelo RGB cada color aparece en sus componentes espectrales primarias: rojo, verde, azul. Este modelo está basado en el sistema de coordenadas cartesianas. El subespacio de color de interés es el tetraedro mostrado en la figura (2.2), en el cual los valores RGB están en tres vértices; cyan, magenta y amarillo se sitúan en otros tres vértices, el negro corresponde al origen y el blanco se sitúa en el vértice más alejado del origen. En este modelo, la escala de grises se extiende desde el negro al blanco a lo largo de la diagonal que une esos dos puntos, y los colores son puntos dentro del tetraedro, definido por vectores desde el origen [3]. Las imágenes en este modelo se forman por la combinación en diferentes proporciones de cada uno de los colores primarios RGB. El modo RGB asigna un valor de intensidad a cada píxel que oscile entre 0 (negro) y 255 (blanco) para cada una de los componentes RGB de una imagen en color. Por ejemplo, un color rojo brillante podría tener un valor R de 246, un valor G de 20 y un valor B de 50. El rojo más brillante que se puede conseguir es el R: 255, G: 0, B: 0. Cuando los valores de los tres componentes son idénticos, se obtiene un matiz de gris. Si el valor de todos los componentes es de 255, el resultado será blanco puro y será negro puro si todos los componentes tienen un valor 0.

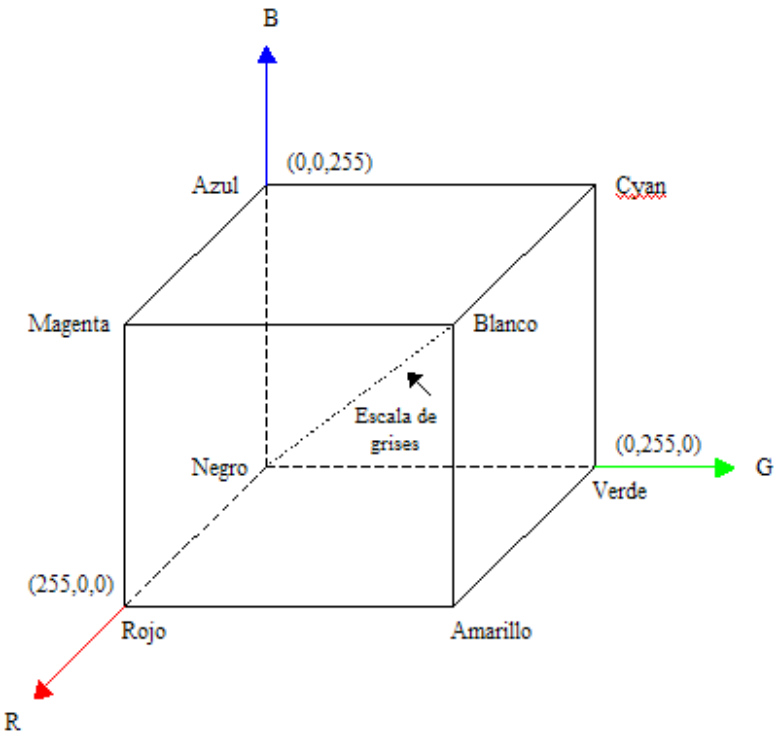


Figura 2.2: Tetraedro de Color RGB

2.1.1.2 El modelo HSI

El matiz es un atributo del color que describe su pureza (p.e. rojo puro), mientras que la saturación proporciona una medida del grado en el que el color puro es diluido con luz blanca. El modelo de color HSI debe su utilidad a dos hechos fundamentales: primero, la componente de intensidad I se puede separar de la información del color en la imagen; segundo, las componentes de matiz y saturación están íntimamente relacionadas con el modo en que los humanos perciben el color. Estas características hacen del modelo HSI una herramienta ideal para desarrollar algoritmos de procesamiento de imágenes basados en alguna de las sensaciones de color del sistema visual humano [3].

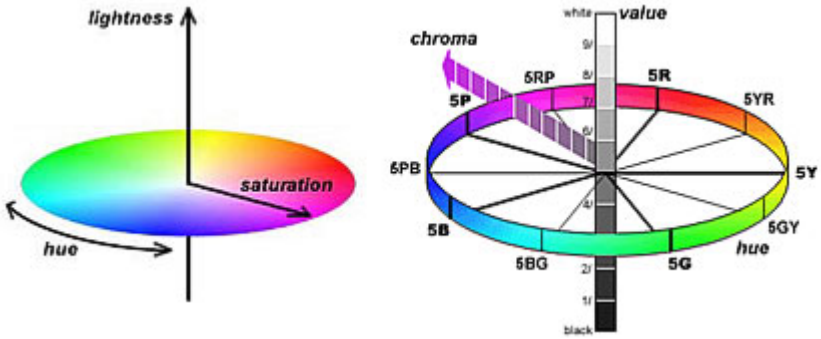


Figura 2.3: El lado izquierdo muestra cómo medir el matiz, la saturación e intensidad. El lado derecho muestra la cromaticidad (chroma) y la intensidad [11]

2.1.2 Conversión de RGB a HSI

El matiz (H) de un punto de color P es el ángulo del vector con respecto al eje rojo. Por tanto, cuando H=0º, el color es rojo, mientras que cuando H=60º, el color es amarillo; con H=120º, el color es verde; con H=180º, el color es cyan; con H=240, el color es azul, y así sucesivamente. La saturación S del punto de color P es el grado de dilución del color blanco y es proporcional a la distancia de P al centro, como se observa en la figura (2.3). A medida que P se aleja del centro y se aproxima a la orilla del círculo cromático, mayor es el grado de saturación del color.

La intensidad en el modelo HSI se mide con respecto a una línea perpendicular al círculo cromático que pasa a través de su centro. Las intensidades a lo largo de esta línea por debajo del círculo cromático progresan hacia el negro y, por encima, hacia el blanco [3].

Combinando matiz, saturación e intensidad en un espacio de color 3D, se obtiene la representación de la figura (2.4). Una vez establecido el nivel de intensidad, se aplican los conceptos explicados anteriormente para determinar el matiz y la saturación. Cualquier punto en la superficie de la figura (2.4) formada por conos representa un color saturado puro. Todo lo anterior es aplicable a puntos dentro de la estructura, la única diferencia es que los colores se hacen menos saturados a medida que se aproximan al eje vertical que une blanco con negro.

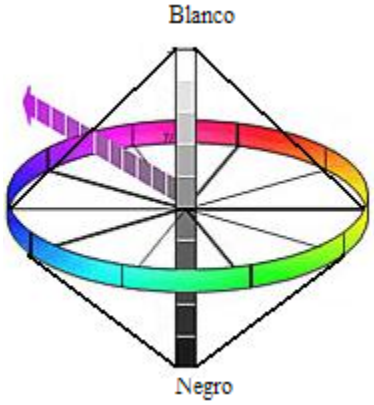


Figura 2.4:Representación del Espacio de Color en 3D de HSI

Los colores en el modelo HSI se definen con respecto a los valores de rojo, verde, azul, dados en términos de los colores primarios RGB por:

$$r = R/(R+G+B) \quad g = G/(R+G+B) \quad b = B/(R+G+B) \tag{2.1}$$

donde:

$$r + g + b = 255 \tag{2.2}$$

Mientras que R, G y B pueden todos ellos tomar el valor 255 simultáneamente, las variables r, g, b deben satisfacer a la ecuación (2.2).

Para cualesquiera tres componentes de color R, G y B, cada una en el rango [0,255], la componente de intensidad en el modelo HSI se define como:

$$I = (1/3)*(R+G+B) \tag{2.3}$$

cuyos valores están en [0,255].

El siguiente paso consiste en obtener H y S. En ambos casos es preciso recurrir a construcciones geométricas un tanto complejas. Finalmente se llega a las siguientes expresiones para H y S en términos de R, G y B:

$$H = \cos^{-1}\left\{\frac{(1/2)[(R-G)+(R-B)]}{\left[(R-G)^2+(R-B)(G-B)\right]^{1/2}}\right\} \tag{2.4}$$

$$S = 1 - \frac{3}{R+G+B}\left[\min(R,G,B)\right] \tag{2.5}$$

Por tanto los valores HSI se obtienen con las ecuaciones (2.3), (2.4) y (2.5), haciendo H=360º-H si B/I > G/I. Finalmente, si S=0, el matiz no está definido y la saturación está indefinida si I=0.

2.1.3 Conversión de HSI a RGB

Para valores de HSI, se trata de encontrar la correspondencia de los valores RGB en el rango de [0,255]. Hay que recordar que el rango del matiz es [0º,360º], el de la saturación es [0,255] y el de la intensidad [0,255].

Para el sector RG (0º < H <= 120º) se tiene:

$$b = \frac{1}{3}(1 - S) \tag{2.6}$$

$$r = \frac{1}{3}\left[1 + \frac{S\cos H}{\cos(60^o - H)}\right] \tag{2.7}$$

$$g = 1 - (r + b) \tag{2.8}$$

Para el sector GB (120º < H <= 240º), se obtiene:

$$H = H - 120^o \tag{2.9}$$

$$r = \frac{1}{3}(1 - S) \tag{2.10}$$

$$g = \frac{1}{3}\left[1 + \frac{S\cos H}{\cos(60^o - H)}\right] \tag{2.11}$$

$$b = 1 - (r + g) \tag{2.12}$$

Para el sector BR (240º < H <= 360º),

$$H = H - 240^o \tag{2.13}$$

$$g = \frac{1}{3}(1 - S) \tag{2.14}$$

$$b = \frac{1}{3}\left[1 + \frac{S\cos H}{\cos(60^o - H)}\right] \tag{2.15}$$

$$r = 1 - (g + b) \tag{2.16}$$

Los valores R, G y B se obtienen a partir de las ecuaciones (2.3), (2.4) y (2.5) teniendo en cuanta las igualdades R=3Ir, G=3Ig y B=3Ib.

2.2 Muestreo

Para realizar el muestreo, se procesan los datos (imágenes) del sistema de visión con operaciones cartesianas básicas y operaciones individuales. Una vez aplicadas dichas operaciones los datos serán tomados con la ayuda de un autómata.

2.2.1 Representación de Imágenes Digitales

El término imagen se refiere a una función de intensidad bidimensional que se representará indistintamente como $f(x, y)$, donde x e y son las coordenadas espaciales y el valor de f en cualquier punto (x, y) es proporcional a la intensidad o nivel de gris de la imagen en ese punto. La siguiente figura muestra la convención utilizada para la representación de imágenes digitales.

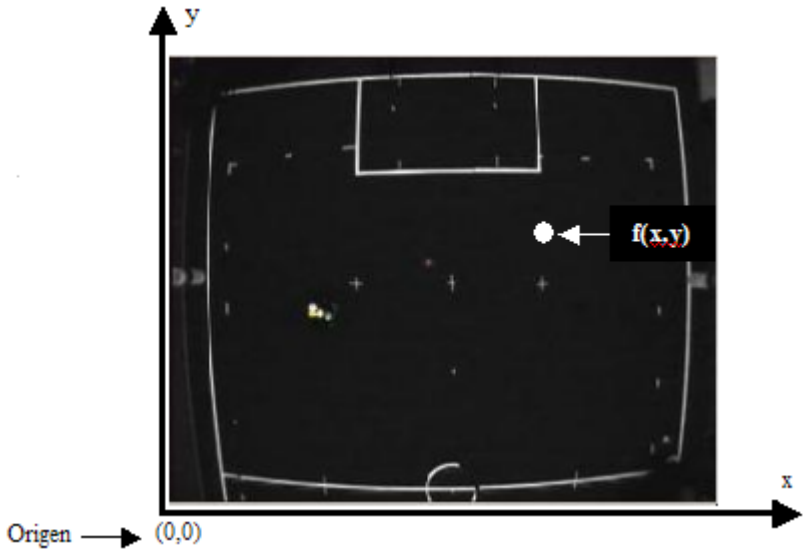


Figura 2.5: Representación de una Imagen Digital

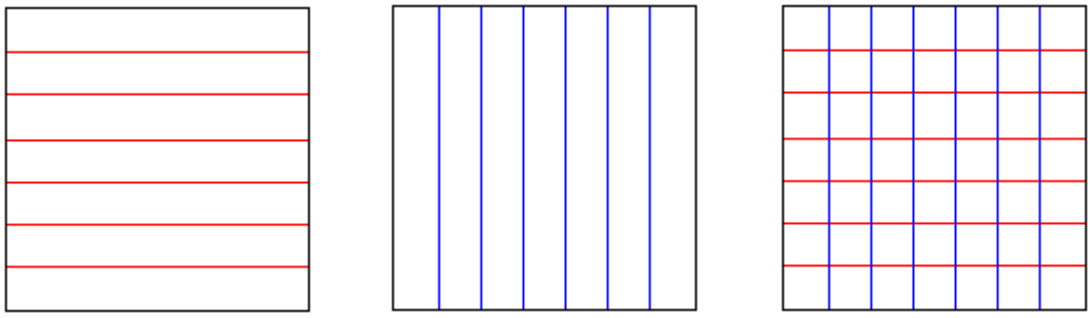
Si se trata de imágenes en escala de gris, $f(x,y)=ng$, donde $0 \le ng \le 255$ es el nivel de gris correspondiente a cada punto (pixel) de la imagen. Para imágenes a color, $f(x,y)=color$, donde *color* contiene tres componentes, estas componentes pueden ser RGB o HSI; donde $0 \le R \le 255$, $0 \le G \le 255$, $0 \le B \le 255$ y $0 \le H \le 360$, $0 \le S \le 255$, $0 \le I \le 255$.

2.2.2 Operaciones Cartesianas Básicas

Una vez que se tiene definido el sistema de ejes cartesianos, ya es posible realizar cualquier tipo de operaciones con base en dichos ejes. Una de tales operaciones, que resulta ser de gran utilidad, es medir la distancia en píxeles entre dos puntos de la imagen de coordenadas $a=(x_0,y_0)$ y $b=(x_1,y_1)$, cuya expresión viene dada por:

$$d(a,b)=\sqrt{(x_0-x_1)^2+(y_0-y_1)^2} \tag{2.17}$$

Otra operación que se puede realizar es la formación de cuadros a través de la intersección de líneas verticales con líneas horizontales. Para formar cuadros con estas líneas se necesita mínimo de dos líneas verticales y de dos líneas horizontales formando así un cuadro, por lo que en general el número de líneas verticales y horizontales deben ser pares.



Las líneas rojas son horizontales | Las líneas azules son verticales | Intersección de las líneas horizontales y verticales

Figura 2.6:Muestra la manera de formar cuadros a partir de líneas horizontales y verticales

2.2.3 Operaciones Individuales

Las operaciones individuales implican la generación de una nueva imagen modificando el valor del píxel en una simple localización basándose en una regla global aplicada a cada localización de la imagen original. El proceso consiste en obtener el valor del píxel de una localización dada en la imagen, modificándolo por una operación lineal o no lineal y colocando el valor del nuevo píxel en la correspondiente localización de la nueva imagen. El proceso se repite para todas y cada una de las localizaciones de los píxeles en la imagen original [3].

2.2.4 Operador Umbral

Esta clase de transformación crea una imagen de salida binaria a partir de una imagen de grises, donde el nivel de transición está dado por el parámetro de entrada p_1 [3].

La función de transformación es la siguiente:

$$q = \begin{cases} 0 \;\; para \;\; p \le p_1 \\ 255 \;\; para \;\; p > p_1 \end{cases} \tag{2.18}$$

2.3 Autómata Finito

Un autómata finito es un modelo matemático de una máquina que acepta cadenas de un lenguaje definido sobre un alfabeto $\sum$. Consiste en un conjunto finito de estados y un conjunto de transiciones entre esos estados, que dependen de los símbolos de la cadena de entrada. El autómata finito acepta una cadena si la secuencia de transiciones correspondientes a los símbolos de la cadena conduce desde el estado inicial a un estado final.

Si para todo estado del autómata existe como máximo una transición definida para cada símbolo del alfabeto, se dice que el autómata es determinístico (AFD).

Formalmente un autómata finito se define como una estructura $A = (\sum, Q, \delta, s, t)$ donde:

$\sum =$ Alfabeto o conjunto finito de símbolos
$Q =$ Conjunto finito de estados
$\delta =$ Función de transición de estados, que se define como $\delta : Q \times \sum \to Q$

$s \in Q$ Estado inicial
$t \in Q$ Estado final

La forma gráfica de representar a cada autómata es un grafo dirigido, llamado diagrama de transición de estados. Cada nodo del grafo corresponde a un estado. El estado inicial se indica mediante una flecha que no tiene nodo origen. Los estados finales se representan con un círculo doble. Si existe una transición del estado $q_i \in Q$ al estado $q_j \in Q$ para un símbolo de entrada $k \in \sum$, existe entonces un arco rotulado desde el nodo $q_i \in Q$ al nodo $q_j \in Q$; es decir que $\delta(q_i, k) = q_j$, se representa en el diagrama

Figura 2.7:Representación Gráfica de un Autómata a través de un Grafo Dirigido

2.4 Toma de Muestras

Para tomar las muestras se debe contar con una imagen que tenga un patrón de rejilla. La imagen se debe convertir al modelo de color HSI o en niveles de grises (solo la I del modelo HSI). Con el operador umbral se binariza la imagen, para encontrar la rejilla con la ayuda de un autómata y la operación cartesiana de intersección de líneas.

El autómata a usar se define de la siguiente manera

$$A = (\{Blanco, Negro, Línea\}, \{Inicio, q, Fin\}, \delta, Inicio, Fin)$$

Descripción de δ			
	Línea	*Blanco*	*Negro*
Inicio	*q*	*Inicio*	*Inicio*
q	-------	*q*	*Fin*
Fin	-------	-------	-------

Tabla 2.1: Transición de Estados del Autómata Reconocedor de Líneas para Interceptar

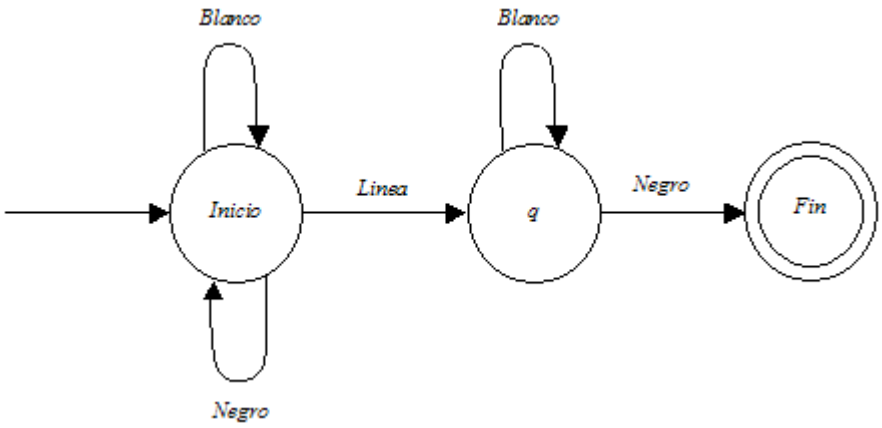


Figura 2.8: Gráfica del Autómata Reconocedor de líneas para interceptar

La *Línea* que se usa, para que el autómata realice la transición del estado *Inicio* al estado *q*, es una "sucesión" de píxeles blancos. Estos píxeles deben existir en un vector sin importar si están separados. El vector puede ser renglón o columna.

Una vez realizada la transición al estado *q*, se guarda la posición del vector con respecto a la matriz que forma la imagen y se coloca en el siguiente vector. La posición que se guarda es la de inicio y fin.

Estando activado el estado *q*, cada que lee el primer píxel Blanco en el vector aumenta en uno la posición de fin y lee el siguiente vector. Si lee todo un vector de píxeles negros entonces el autómata termina.

2.5 Algoritmo para la detección de cuadros en patrón de rejilla

-Imagen con un patrón de rejilla

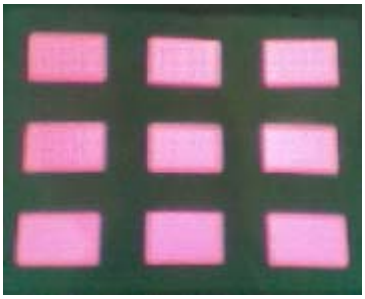


Figura 2.9:Imagen con un Patrón de Rejilla

-Se obtiene la imagen en una representación de matriz.

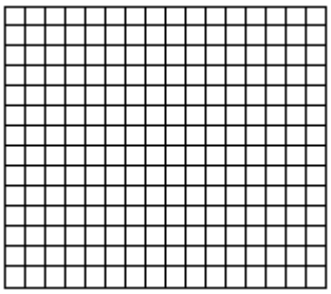


Figura 2.10:Imagen en una Representación de Matriz

-Se da la cantidad de puntos para definir una línea.

línea: 10 píxeles blancos en un vector de la imagen

Línea Horizontal

Línea Vertical

Figura 2.11:Ejemplo de la Definición de "Línea"

-Se da el nivel de gris mínimo para binarizar la imagen y buscar el patrón

$$imagen(x,y)=\begin{cases}0 \;\; para \;\; p \le nivel\_gris \\ 255 \;\; para \;\; p > nivel\_gris\end{cases}$$

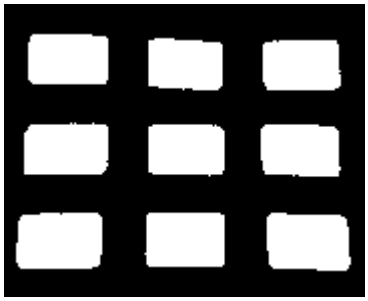


Figura 2.12:Imagen Binarizada del Patrón de Rejilla

-Se activa el autómata en forma vertical y horizontal

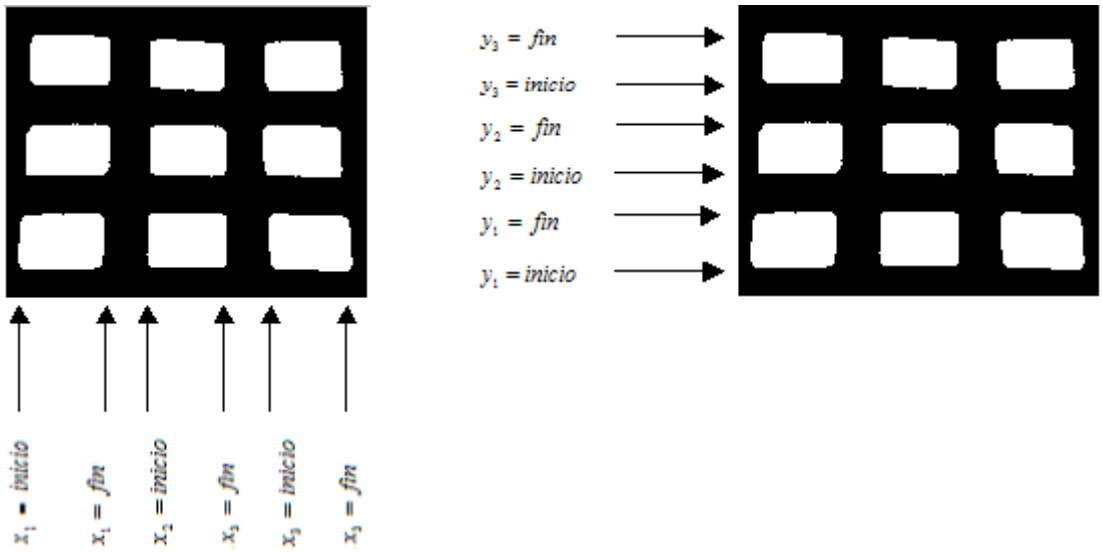


Figura 2.13:Muestra la forma en que el autómata encuentra las intersecciones

- Con las coordenadas obtenidas formamos cuadros que contienen las muestras deseadas

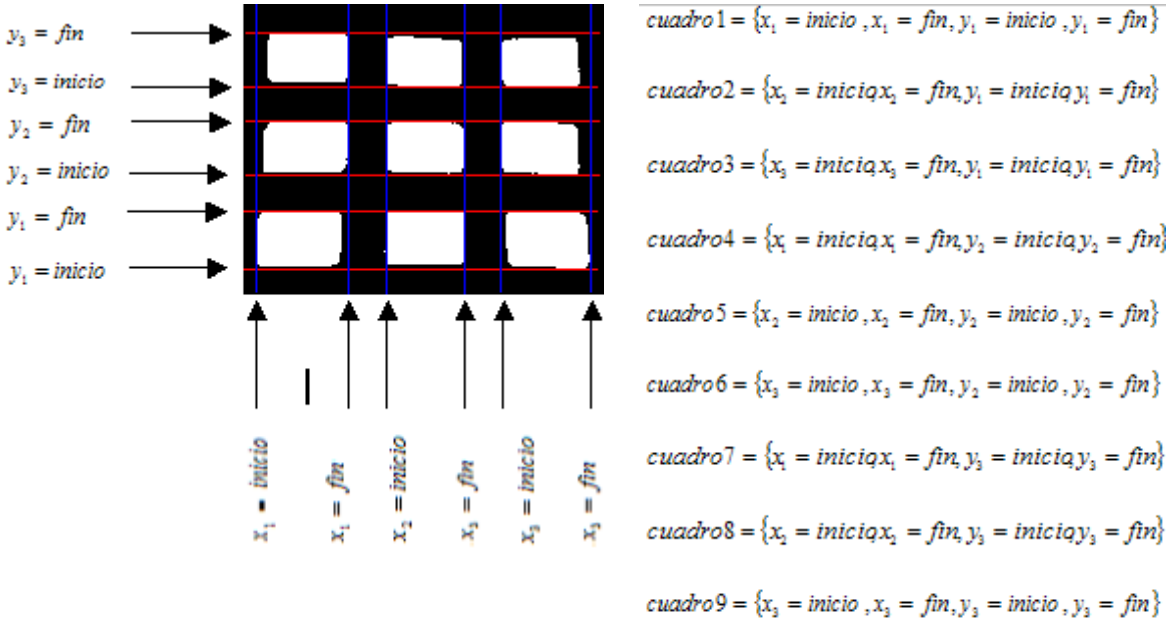


Figura 2.14:Muestra cómo se forman los cuadros a partir de las coordenadas regresadas por el autómata

- De cada cuadro tomamos las muestras que cumplan con el nivel de gris deseado. El dato que se guarda es el que se quiere utilizar como muestra. Ya que con las coordenadas podemos mapear a la imagen original y podemos obtener datos R,G,B,H,S,I.

si $cuadro(x, y) > nivel\_gris$ entonces guarda la muestra

2.6 Cuantificación Vectorial

Al aplicar la cuantificación se obtiene compresión, puesto que el conjunto de valores posibles de salida es menor al de entrada. En contrapartida se introduce un error ya que es un proceso irreversible (a partir de la señal cuantificada no se puede recuperar la señal original).

Existe una relación entre la compresión y el error introducido por la cuantificación. Cuanto mayor sea el conjunto posible de valores de salida, más parecidos serán las señales previa y posterior a la cuantificación, pero la compresión será menor.

La cuantificación vectorial consistente en tratar conjuntamente grupos de N muestras, siendo N la dimensión de los vectores. La cuantificación vectorial trabaja sobre vectores de entrada N-dimensiónales en el espacio R^N. Todo el conjunto de vectores N-dimensionales posibles de entrada sólo permite K vectores posibles a la salida.

En la cuantificación vectorial se utiliza un libro de códigos, conocido como codebook. El codebook es una tabla que contiene los vectores posibles a la salida del cuantificador. Cada vector del codebook se llama codevector y su correspondiente índice en la tabla codeword [4].

Al igual que en la cuantificación escalar, para obtener una representación eficiente, se almacena el índice de la tabla para cada una de las muestras de la señal de entrada y la tabla que permite obtener el correspondiente valor de salida [4].

En el diseño de un cuantificador vectorial hay que escoger el número de bits del codebook (o equivalentemente el número de codewords = $2^{\text{No. Bits de codebook}}$), la dimensión de los vectores de entrada, y los valores de los codevectores para que la cuantificación implique el mínimo error posible de la señal cuantificada respecto a la original [4].

2.6.1 Proceso de cuantificación

Cuantificar un vector de entrada (x) consiste en encontrar su "vecino más cercano" (y_j) dentro del codebook. Para ello hay que calcular su distancia a todos los vectores del codebook y escoger el que proporcione un valor menor. Es necesario disponer de una medida de distancia. Por ejemplo, la distancia euclídea entre vectores.

$$d\left(\vec{x}, \vec{y_j}\right) = \sum_{i=1}^{N} \left(x_j - y_{ji}\right)^2 = D_j \tag{2.19}$$

con $\vec{x} = [x_1, x_2, ..., x_N]$ e $\vec{y_j} = [y_{j1}, y_{j2}, ..., y_{jN}]$.

El algoritmo es el siguiente [4]:

1. Inicializar la distancia mínima d igual a la distancia inicial D_1, y los contadores j e i (índice del codevector más cercano):

$$d = D_1; \qquad j = 1; \qquad i = 1$$

2. $D_{j+1} = d\left(\vec{x}, \vec{y}_{j+1}\right)$

3. Si $D_{j+1} < d \Rightarrow \begin{cases} d = D_{j+1} \\ i = j+1 \end{cases}$

4. Si $j < K \Rightarrow \begin{cases} j = j+1 \\ volver\ al\ \ punto\ \ 2 \end{cases}$

   Si $j = K$, el resultado es el índice i y la distancia mínima d.

La obtención del codebook suele realizarse en 2 pasos

- Obtención de un codebook inicial
- Mejora del codebook inicial mediante el algoritmo de Lloyd

2.6.2 Obtención del codebook inicial

Es importante obtener un buen codebook inicial, para conseguir un resultado final satisfactorio. La creación del codebook inicial y su mejora se realiza a partir de una secuencia de entrenamiento (SE), formado por un conjunto de vectores pertenecientes al espacio N-dimensional.

Por tanto, el codebook estaría especialmente adaptado a la secuencia de entrenamiento, puesto que se obtendrá a partir de él. Para conseguir una cuantificación eficiente de vectores no contenidos en la secuencia de entrenamiento, dicha secuencia debe ser lo más representativa posible de la distribución de la señal de entrada. En caso contrario, el codebook se especializará en unos vectores determinados a costa de penalizar otros [4].

2.6.3 Método de Splitting

Permite crear codebooks en potencia de 2. Esto es debido a que se trata de un proceso iterativo en el que en cada iteración se dobla el tamaño del codebook obtenido en la iteración anterior.

El algoritmo consta de los siguientes pasos:

1. Se crea un codebook con un único vector $\vec{c}_0$ que es el centroide (codevector) de toda la secuencia de entrenamiento L.

$$\vec{c}_0 = \frac{1}{L}\sum_{i=1}^{L}\vec{v}_i \tag{2.20}$$

   donde la secuencia de entrenamiento (SE) es el conjunto de L vectores

$$SE = \left\{\vec{v}_1, \vec{v}_2, ..., \vec{v}_L\right\} \tag{2.21}$$

2. Se divide el centroide en dos codevectores, obteniendo un codebook de dos vectores:

$$codebook = \left\{\vec{c}_0, \vec{c}_0 + \vec{\varepsilon}\right\} \tag{2.22}$$

   donde ε es una pequeña perturbación. Por ejemplo, puede hacerse la componente i-ésima de ε proporcional a la desviación estándar de la componente i-ésima del conjunto de vectores de la SE.

$$\begin{aligned}
\vec{v}_1 &= \{v_{11}, v_{12}, ..., v_{1N}\} \\
\vec{v}_2 &= \{v_{21}, v_{22}, ..., v_{2N}\} \\
&\vdots \\
\vec{v}_L &= \{v_{L1}, v_{L2}, ..., v_{LN}\}
\end{aligned} \tag{2.23}$$

$$\vec{\varepsilon} = \{\sigma_1, \sigma_2, ..., \sigma_N\}$$

con

$$\begin{aligned} \sigma_1 &= desviación\ estándar\{v_{11}, v_{21}, ..., v_{L1}\} \\ \sigma_2 &= desviación\ estándar\{v_{12}, v_{22}, ..., v_{L2}\} \\ &\vdots \\ \sigma_N &= desviación\ estándar\{v_{1N}, v_{2N}, ..., v_{LN}\} \end{aligned} \tag{2.24}$$

3. Se aplica el algoritmo de mejora de codebooks (Iteración de Lloyd)

4. Se vuelve a dividir en dos y mejorar cada uno de los codevectores del codebook hasta llegar al tamaño deseado, que será potencia de dos.

El cálculo de la perturbación ε_i usada en un determinado vector c_i se puede realizar a partir del cálculo de la desviación estándar de las componentes de los vectores de la SE asignados a dicho vector c_i. De esta forma cuanto mayor sea la dispersión (o desviación estándar) de la nube de vectores de entrenamiento asignada al codeword c_i, más separados estarán los vectores resultantes c_i y $c_i + \varepsilon_i$.

La ventaja de éste método es que si se almacenan en memoria los diferentes codebooks previos a la división, será posible realizar una búsqueda de codewords en una estructura de árbol binario, la cual requiere menos búsqueda.

2.6.4 Iteración de Lloyd

Dado un codebook inicial, puede mejorarse mediante el siguiente proceso:

1. Dividir la SE en sectores (clusters) usando la condición de vecino más cercano. De esta forma, cada uno de los codewords tendrá asignado un subconjunto de vectores pertenecientes a la SE [4].

2. Una vez obtenidas las particiones, el mejor codevector será el promedio de todos los vectores que tiene asignados. Por tanto, podemos calcularlo y sustituir el antiguo codevector. Si en el primer paso hemos obtenido una región vacía, se divide en 2 el centroide que más contribuya a la distorsión total del codebook [4].

Para la implementación de la iteración de Lloyd se necesita:

- La SE formada por L vectores de dimensión N

- El codebook de k vectores de dimensión N
- Una tabla intermedia formada por k filas de N+1 posiciones

El procedimiento a seguir es el siguiente:

Para cada uno de los L vectores de la SE buscamos su vecino más cercano en el codebook y en la correspondiente posición de la tabla intermedia, inicializada previamente a cero en todas sus casillas, sumamos el vector de la SE en las correspondientes N primeras casillas. La posición N+1 es un contador con el número de vectores asociados a dicha partición. Por tanto, su contenido debe incrementarse en 1 unidad. Se hace de esta forma con todos los vectores de la SE, de manera que al final en la tabla intermedia tenemos el valor acumulado de todos los vectores asociados a una determinada partición y el número de vectores asignados. Por tanto, el paso final es actualizar el codebook sustituyendo cada uno de las k filas por las correspondientes N primeras casillas de la tabla intermedia, divididas por la casilla N+1 [4].

2.6.5 Algoritmo de Lloyd Generalizado

Tras calcular el codebook es posible que las particiones de la SE hayan cambiado, con lo cual es posible recalcularlas y encontrar unos nuevos codevectores que reduzcan más la distorsión, y nuevamente varíen las particiones. Por tanto, es necesario realizar el proceso de forma iterativa.

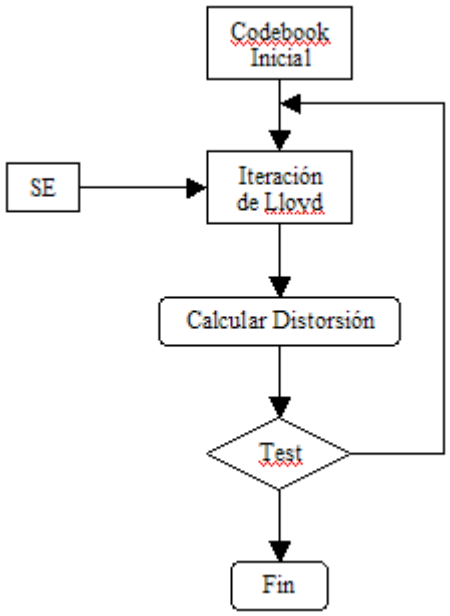


Figura 2.15: Diagrama de Flujo del Algoritmo de Lloyd Generalizado

Normalmente, al cabo de unas cuantas iteraciones se estabiliza, y la aplicación de la iteración de Lloyd no produce una mejora significativa. Por lo tanto, se repite todo el proceso hasta que la reducción de la distorsión entre 2 iteraciones consecutivas sea nula o poco significativa [4].

2.7 Redes Neuronales Artificiales

Una neurona biológica es una célula especializada en procesar información. Está compuesta por el cuerpo de la célula (soma) y dos tipos de ramificaciones: el axón y las dendritas. La neurona recibe señales (impulsos) de otras neuronas a través de sus dendritas y transmite señales generadas por el cuerpo de la célula a través del axón.

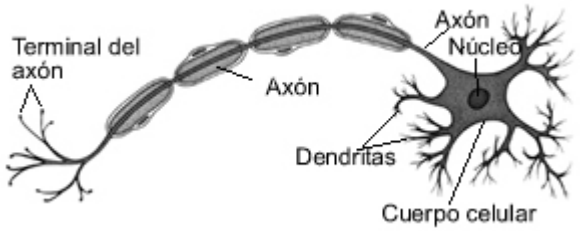


Figura 2.16: Estructura de una Neurona [31]

Las neuronas se comunican entre sí mediante trenes de pulsos de corta duración (del orden de milisegundos). El mensaje, está modulado en la frecuencia de transmisión de los pulsos. Esta frecuencia varía sobre los 100Hz. Por tanto, es más de un millón de veces inferior a la velocidad de conmutación de los circuitos electrónicos típicos de nuestros días.

Sin embargo, el ser humano es capaz de realizar tareas complejas en un tiempo inferior y con un porcentaje de aciertos superior al conseguido actualmente mediante ordenadores. Por ejemplo, tareas de reconocimiento del habla, locutor a través de su voz, cara, etc. Ello es debido a la paralelización masiva que realiza el cerebro, que está extremadamente lejos de poder ser implementada en un circuito electrónico.

Las redes neuronales artificiales pretenden emular la red neuronal biológica y se obtienen interconectando neuronas. Para ello se supone un modelo matemático simplificado de la neurona biológica.

2.7.1 Modelos de Redes Neuronales Artificiales (RNA)

2.7.1.1 Componentes básicos de las redes neuronales

Una red neuronal es un paradigma general computado matemáticamente que modela las operaciones de sistemas neuronales. En 1943, McCulloch, un neurobiólogo, y Pitts, un estadístico, publicaron un artículo titulado "A logical calculus of ideas imminent in nervous activity". Este artículo inspiró el desarrollo de la computación moderna digital, o el cerebro electrónico, como lo llamó John Von Neumann. Aproximadamente al mismo tiempo, Frank Rosenblatt fue también motivado por este artículo a investigar la computación del ojo, el cual eventualmente permite la primer generación de la red neuronal, conocida como perceptrón. A continuación se da una pequeña revisión de modelos de RNA con el propósito de tener conceptos básicos de los modelos de redes neuronales [8].

2.7.1.1.1 Modelo Neuronal de McCulloch y Pitts

Entre numerosos modelos de redes neuronales que han sido propuestos sobre los años, todos comparten un bloque común de construcción conocido como una neurona y una estructura de interconexión de red. El modelo de neurona mas usado esta basado en el trabajo de McCulloch y Pitts y es ilustrado en la siguiente figura (2.17).

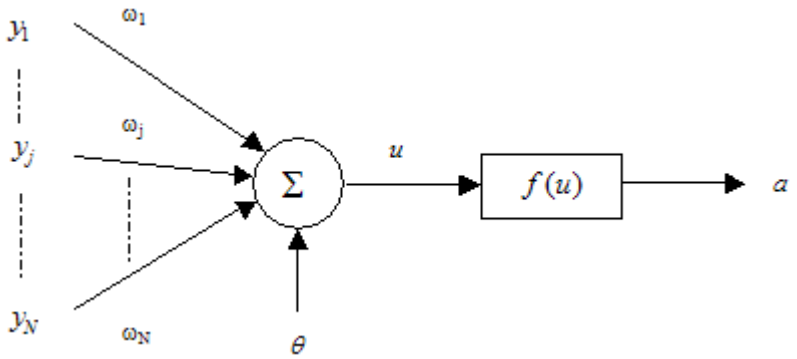


Figura 2.17: Neurona Basada en el Modelo de McCulloch y Pitts

En la figura (2.17), cada neurona consiste de dos partes: la función de red y la función de activación. La función de red determina como las entradas de la red $\{y_j; 1 \le j \le N\}$ son combinadas dentro de la neurona. En esta figura, una combinación lineal de pesos es adoptada:

$$u = \sum_{j=1}^{N} w_j y_j + \theta \tag{2.25}$$

$\{w_j; 1 \le j \le N\}$ son parámetros conocidos como pesos sinápticos. La cantidad θ es llamada bias (o umbral) y es usada para modelar el umbral.

La salida de la neurona, denotada por a_i en la figura (2.17), está relacionada a la entrada de la red u_i vía una transformación lineal o no lineal llamada la función de activación:

$$a = f(u) \tag{2.26}$$

En varios modelos de redes neuronales, diferentes funciones de activación se han propuesto. Las funciones de activación más usadas comúnmente están resumidas en la tabla (2.2).

Tabla 2.2: Función de Activación de Neuronas

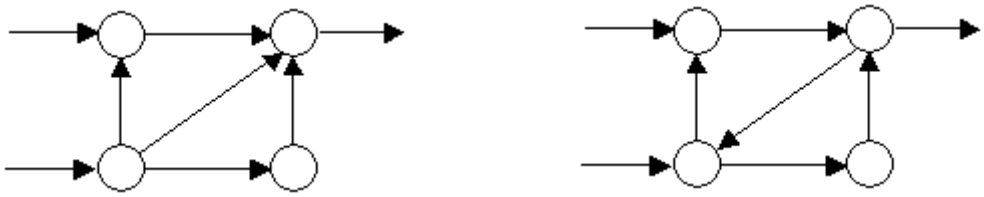
Función de activación	Fórmula $a = f(u)$	Derivada $df(u)/du$	Comentario
Sigmoidea	$f(u) = \frac{1}{1+e^{-u/T}}$	$f(u)[1-f(u)]/T$	Comúnmente usada; la derivada puede ser computada desde *f(u)* directamente
Tangente hiperbólica	$f(u) = \tanh\left(\frac{u}{T}\right)$	$\left(1-[f(u)]^2\right)/T$	*T* = parámetro de temperatura
Lineal	$f(u) = au + b$	*A*	

En la tabla (2.2) se listan tanto las funciones de activación como sus derivadas. En las funciones de activación hiperbólica y sigmoidea, sus derivadas pueden ser computadas directamente desde el conocimiento de $f(u)$.

2.7.1.1.2 Topología de la red neuronal

En una red neuronal, múltiples neuronas son interconectadas para formar una red y así facilitar la computación distribuida. La configuración de las interconexiones pueden ser descritas eficientemente con un grafo dirigido. Un grafo dirigido consiste de nodos (en el caso de una red neuronal, neuronas, y también como entradas externas) y arcos directos (en el caso de una red neuronal, conexiones sinápticas) [8].

La topología del grafo puede ser categorizada como acíclica o cíclica. En la figura (2.18a); una red neuronal con topología acíclica consiste de no tener ciclos que alimenten hacia atrás. Tal red neuronal acíclica es frecuentemente usada para aproximar un mapeo no lineal entre sus entradas y sus salidas. Como se muestra en la figura (2.18b), una red neuronal de topología cíclica contiene al menos un ciclo formado por arcos dirigidos. Tal red neuronal es también conocida como una red recurrente. Debido al ciclo de retroalimentación, una red recurrente permite un modelo de sistema dinámico no lineal que contiene memoria interna. Las redes neuronales recurrentes frecuentemente exhiben comportamientos complejos y continúan un tópico de investigación continua en el campo de las redes neuronales artificiales [8].



(a)Topología acíclica *(b) Topología cíclica*

Figura 2.18: Ilustración de (a) un grafo acíclico y (b)un grafo cíclico

2.7.1.2 Modelo del Perceptrón Multicapa (PMC)

El perceptrón multicapa es por mucho la red neuronal mas conocida y popular entre todos los paradigmas de redes neuronales existentes. Para introducir el perceptrón multicapa, primero se discutirá el modelo del perceptrón.

2.7.1.2.1 Modelo del Perceptrón

Un perceptrón multicapa es una variante del modelo del perceptrón original propuesto por Rosenblatt en la década de 1950. En el modelo del perceptrón, se emplea una sola neurona con una función de red lineal de pesos y una función de activación de umbral. La entrada a esta neurona $\vec{x} = (x_1, x_2, ..., x_n)$ es un vector descriptivo en un

espacio descriptivo n-dimensional. La función de la red $u(\vec{x})$ es la suma de los pesos de las entradas [8]:

$$u(\vec{x}) = w_0 + \sum_{i=1}^{n} w_i x_i \tag{2.27}$$

y la salida $y(\vec{x})$ es obtenida desde $u(\vec{x})$ vía la función de activación del umbral:

$$y(\vec{x}) = \begin{cases} 1 & u(\vec{x}) \geq 0 \\ 0 & u(\vec{x}) < 0 \end{cases} \tag{2.28}$$

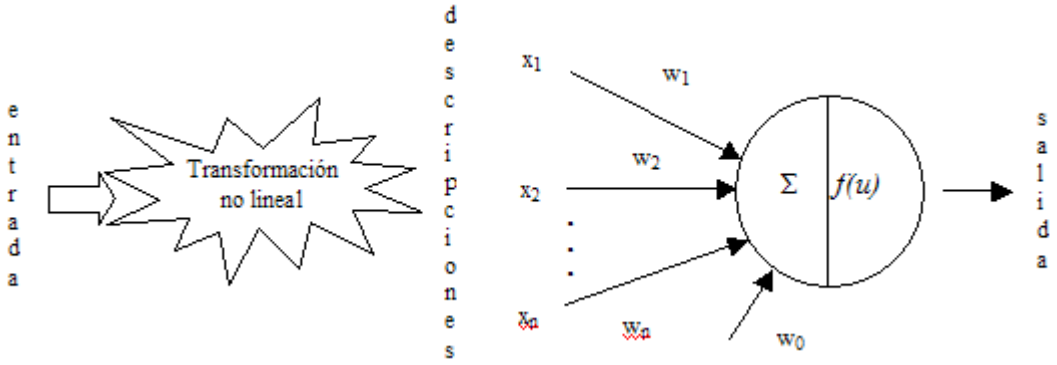


Figura 2.19 Modelo de la red neuronal del perceptrón

El modelo de neurona del perceptrón puede ser usado para detección. Por ejemplo, el vector de pesos $\vec{w} = (w_1, w_2, ..., w_n)$ podría representar el patrón de un cierto blanco. Si el vector descriptivo de la entrada $\vec{x}$ es muy semejante a $\vec{w}$ tal que su producto interior excede un umbral $-w_0$, entonces la salida llegará a ser $+1$, indicando la detección de un blanco.

El vector de pesos $\vec{w}$ necesita ser determinado para aplicar el modelo de perceptrón. Frecuentemente, un conjunto de muestras de entrenamiento $\{(\vec{x}(i), d(i)); i \in I_r\}$ y muestras de prueba $\{(\vec{x}(i), d(i)); i \in I_t\}$ son dadas. Aquí $d(i) = (\in \{0,1\})$ es el valor de salida deseado de $y(\vec{x}(i))$ si el vector de pesos $\vec{w}$ es escogido correctamente, e I_r e I_t son índices de conjuntos disjuntos. Un algoritmo de aprendizaje del perceptrón en línea secuencial puede ser aplicado para estimar iterativamente el valor correcto de $\vec{w}$ presentando las muestras de entrenamiento a la neurona del perceptrón en un orden aleatorio y secuencial. El algoritmo de aprendizaje tiene la siguiente formulación [8]:

$$\vec{w}(k+1) = \vec{w}(k) + \eta(d(k) - y(k))\vec{x}(k) \tag{2.29}$$

Donde $y(k)$ es computada usando la ecuación (2.27) y (2.28). En la ecuación (2.29), el factor de aprendizaje $\eta\left(0 < \eta < 1/|\vec{x}(k)|_{\max}\right)$ es un parámetro escogido por el usuario, donde $|\vec{x}(k)|_{\max}$ es la magnitud máxima de las muestras de entrenamiento $\{\vec{x}(k)\}$. El índice *k* es usado para indicar que las muestras de entrenamiento son aplicadas secuencialmente al perceptrón en un orden aleatorio. Cada vez que las muestras de entrenamiento son aplicadas, la salida correspondiente del perceptrón $y(k)$ se usa para ser comparada con la salida deseada $d(k)$. Si ellas están iguales, significa que el vector de pesos $\vec{w}$ es correcto para sus muestras de entrenamiento, los pesos permanecerán sin cambio. Por otra parte, si $y(k) \neq d(k)$, entonces $\vec{w}$ será actualizado con un pequeño paso a lo largo de la dirección del vector de entrada $\vec{x}(k)$. Se ha probado que si las muestras de entrenamiento son linealmente separables, el algoritmo de aprendizaje del perceptrón convergerá a una solución factible del vector de pesos con un número finito de iteraciones. Por otra parte, si las muestras de entrenamiento no son linealmente separables, el algoritmo no convergerá con un determinado valor de η no cero [8].

2.7.1.2.1.1 Aplicaciones del modelo neuronal del perceptrón

Hay demasiadas dificultades en aplicar el modelo neuronal del perceptrón para resolver problemas de detección de señales y clasificación de patrones del mundo real:

1. La transformación no lineal que extrae la apropiada descripción del vector *x* no está especificada.

2. El algoritmo de aprendizaje del perceptrón no convergerá para un valor determinado del factor de aprendizaje η si la descripción de los patrones de entrenamiento no son linealmente separables.

3. A pesar de que la descripción de los patrones sea linealmente separable, no se sabe cuanto se tarde el algoritmo en converger al vector de peso que corresponda al hiperplano que separa la descripción de los patrones.

2.7.1.2.2 Perceptrón Multicapa

Un modelo neuronal del perceptrón multicapa consiste de una alimentación hacia adelante (feed-forward), en las capas de la red de las neuronas de McCulloch y Pittts. Cada neurona en un PMC tiene una función de activación no lineal que frecuentemente

es una diferenciable continua. Algunas funciones de activación más usadas para un PMC incluyen la función sigmoidea y la función tangente hiperbólica [8].

Una configuración típica del PMC es mostrada en la figura (2.20). Cada círculo representa una neurona individual. Estas neuronas están organizadas en capas, clasificadas como la capa oculta 1, la capa oculta 2, y la capa de salida. Mientras las entradas en el fondo son clasificadas como capa de entrada, usualmente no hay un modelo de neurona implementado en esta capa. El nombre de las capas ocultas refiere a el hecho de que la salida de estas neuronas estarán alimentando a las capas neuronales de arriba, por eso, es oculta para el usuario quien solamente observa la salida de las neuronas de la capa de salida. La figura (2.20) ilustra una configuración popular del PMC donde las interconexiones son provistas solamente entre neuronas de capas sucesivas en la red. En la práctica, cualquier interconexión acíclica entre neuronas es permitida.

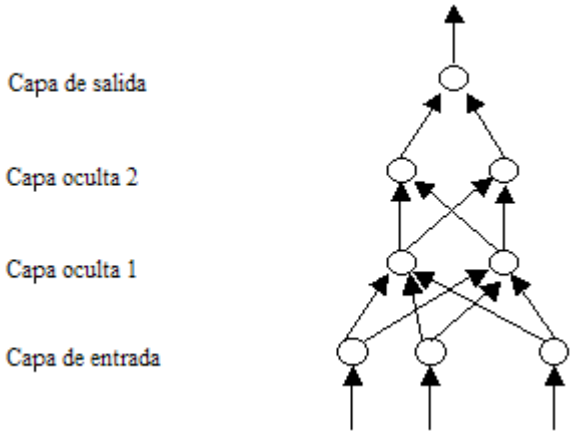


Figura 2.20: Configuración Típica del PMC

Un PMC provee un mapeo no lineal entre sus entradas y salidas.

Se ha probado que con un número suficiente de neuronas ocultas, un PMC con tan pocas como 2 capas de neuronas ocultas, es capaz de aproximar un mapeo complejo arbitrariamente con un soporte finito [8].

2.7.2 Aprendizaje

Una vez fijada la arquitectura y el número de neuronas de la red, hay que calcular el valor de los pesos para que realice la función deseada. Usualmente, la red debe aprender los pesos de las conexiones a partir de un conjunto de patrones de

entrenamiento. Por tanto, no se calculan los pesos de forma analítica sino que es necesario un proceso de aprendizaje automático a partir de una serie de ejemplos.

En vez de fijar una serie de reglas explícitas, es la red neuronal la que aprende las reglas implícitas que relacionan la salida con la entrada, a partir de una colección de ejemplos representativos. Ésta es una de las mayores ventajas de las redes neuronales sobre los sistemas tradicionales.

Para diseñar un proceso de aprendizaje será necesario disponer de:

- Un conjunto de información disponible para el aprendizaje.
- Un proceso para actualizar los pesos, o algoritmo de aprendizaje.

También en este proceso de aprendizaje se debe definir el método de aprendizaje. A continuación se mencionan tres métodos básicos de aprendizaje:

2.7.2.1 Aprendizaje supervisado

La red conoce la salida correcta para cada patrón de entrada. Por tanto, se ajustan los pesos de forma que las salidas de la red sean lo más parecidas posibles a la salida correcta.

2.7.2.2 Aprendizaje por refuerzo

Es una variante del método anterior, en el cual la red únicamente conoce una crítica sobre la exactitud de las salidas, pero no la respuesta correcta.

Utiliza un mecanismo de penalización y recompensa, mediante el cual la red resulta recompensada por una decisión correcta y castigada por una equivocada. De esta forma, el sistema tiende a actuar de la forma favorecida y debilita la tendencia a proporcionar la respuesta penalizada.

2.7.2.3 Aprendizaje autoorganizado

No requiere el conocimiento de la salida correcta, y debe aprender la estructura implícita en los datos. Generalmente requiere un número mayor de datos de entrenamiento.

2.7.3 El error de entrenamiento de retro propagación del PMC

Un paso clave en aplicar el modelo PMC es escoger la matriz de pesos. Asumiendo una estructura PMC de capas, los pesos alimentan a cada capa de neuronas desde una matriz de peso de esa capa (la capa de entrada no tiene una matriz de pesos ya

que no contiene neuronas). Los valores de estos pesos son encontrados usando el método de entrenamiento del error retro propagado (back-propagation).

2.7.3.1 Encontrando los pesos de una simple neurona PMC

Por conveniencia, se considera primero un ejemplo simple consistiendo de una simple neurona para ilustrar este procedimiento. Para claridad de la explicación, la figura (2.21) representa la neurona en 2 partes separadas: una unidad de suma para computar las funciones de red *u*, y un función de activación no lineal $z = f(u)$ [8].

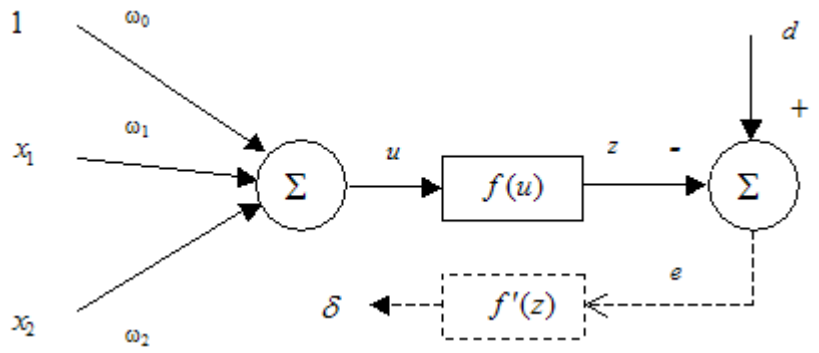


Figura 2.21:Ejemplo del entrenamiento de un PMC por retro propagación, caso de una neurona [8]

La salida de *z* es para ser comparada con un valor deseado del blanco *d*, y sus diferencias, el error $e = d - z$, será computado. Hay 2 entradas $[x_1 \; x_2]$ con pesos correspondientes w_1 y w_2. La entrada clasificada con una constante 1 representa el término bias θ mostrado en la figura (2.7). Aquí, el término bias es clasificado como w_0. La función de red es computada como:

$$u = \sum_{i=0}^{2} w_i x_i = W_X \tag{2.30}$$

donde $x_0 = 1$, $W = [w_0 \; w_1 \; w_2]$ es la matriz de pesos, y $x = [1 \; x_1 \; x_2]^T$ es el vector de entrada.

Dando un conjunto de muestras de entrenamiento $\{(x(k), d(k)); 1 \le k \le K\}$, el error de entrenamiento de retro propagación empieza por alimentar todas las *K* entradas

a través de la red PMC y computando la salida correspondiente $\{z(k); 1 \le k \le K\}$. Aquí se usa una adivinación inicial para la matriz de pesos *W*. Entonces la suma del error cuadrático será computada como:

$$E = \sum_{k=1}^{K} [e(k)]^2 = \sum_{k=1}^{K} [d(k) - z(k)]^2 = \sum_{k=1}^{K} [d(k) - f(W_X(k))]^2 \qquad (2.31)$$

El objetivo es ajustar la matriz de pesos *W* para minimizar el error *E*. Esto permite optimizar el problema no lineal con mínimos cuadrados. Hay varios algoritmos de optimización no lineales disponibles para resolver este problema. Básicamente, estos algoritmos adoptan una formulación iterativa similar:

$$W(t+1) = W(t) + \Delta W(t) \qquad (2.32)$$

donde $\Delta W(t)$ es la corrección hecha a los pesos actuales $W(t)$. Diferentes algoritmos difieren en la forma de $\Delta W(t)$.

Tabla 2.3: Algoritmo de Optimización No Lineal Iterativo para Resolver los pesos del PMC

Algoritmo	$\Delta W(t)$	Comentario
Método del Gradiente Descendente angulado	$= -\eta g(t) = -\eta\, dE/dW$	*g* es conocida como el vector gradiente. η es el tamaño del paso o factor de aprendizaje. Esto es también conocido como el aprendizaje de retro-propagación del error.

Esta sección se enfoca en los pasos del método del gradiente descendente que es también la base del algoritmo de aprendizaje de retro propagación del error. La derivada de la cantidad escalar *E* con respecto a los pesos individuales puede ser computada como sigue [8]:

$$\frac{\partial E}{\partial w_i} = \sum_{k=1}^{K} \frac{\partial [e(k)]^2}{\partial w_i} = \sum_{k=1}^{K} 2[d(k) - z(k)] \left(-\frac{\partial z(k)}{\partial w_i} \right) \quad \text{para } i=0,1,2 \qquad (2.33)$$

donde

$$\frac{\partial z(k)}{\partial w_i} = \frac{\partial f(u)}{\partial u} \frac{\partial u}{\partial w_i} = f'(u) \frac{\partial}{\partial w_i} \left(\sum_{j=0}^{2} w_j x_j \right) = f'(u) x_i \qquad (2.34)$$

De aquí,

$$\frac{\partial E}{\partial w_i} = -2\sum_{k=1}^{K}\left[d(k) - z(k)\right]f'(u(k))x_i(k) \tag{2.35}$$

Con $\delta(k) = \left[d(k) - z(k)\right]f'(u(k))$, la ecuación puede ser expresada como:

$$\frac{\partial E}{\partial w_i} = -2\sum_{k=1}^{K}\partial(k)x_i(k) \tag{2.36}$$

$\partial(k)$ es la señal de error $e(k) = d(k) - z(k)$ modulado por la derivada de la función de activación $f'(u(k))$ y de aquí representa la cantidad de corrección necesaria para ser aplicada a los pesos w_i para la entrada dada $x_i(k)$. El cambio total Δw_i es la suma de cada contribución sobre todas las *K* muestras de entrenamiento. Por eso, la fórmula de actualización de pesos tiene la forma de:

$$w_i(t+1) = w_i(t) + \eta\sum_{k=1}^{K}\delta(k)x_i(k) \tag{2.37}$$

Si una función de activación sigmoidea es usada, entonces $\delta(k)$ puede ser computada como:

$$\delta(k) = \frac{\partial E}{\partial u} = \left[d(k) - z(k)\right]\cdot z(k)\cdot\left[1 - z(k)\right] \tag{2.38}$$

Note que la derivada $f'(u)$ puede ser evaluada exactamente sin alguna aproximación. Cada vez que los pesos son actualizados es llamada una época. En este ejemplo, las *K* muestras de entrenamiento son aplicadas para actualizar los pesos una vez. Así, se dice que la época es de tamaño *K*. En la práctica, el tamaño de la época puede variar entre uno y el total del número de muestras.

2.7.3.2 La retro-propagación del error en un perceptrón multicapa

Hasta ahora, se ha discutido como ajustar los pesos (entrenamiento) de un PMC con una sola capa de neuronas. Esta sección discute como ejecutar el entrenamiento para un PMC multicapa. Primero, se adoptan nuevas notaciones para distinguir neuronas en diferentes capas. En la figura (2.22) la función de red y la salida correspondiente a la k-ésima muestra de entrenamiento de la neurona j-ésima de la $(L-1)$-ésima son denotados por $u_j^{L-1}(k)$ y $z_j^{L-1}(k)$, respectivamente. La capa de entrada es la capa cero. En

particular, $z_j^0(k) = x_j(k)$. La salida es alimentada en la neurona i-ésima de la capa L-ésima vía un peso sináptico denotado por $w_{ij}^L(t)$ o, por simplicidad, w_{ij}^L, desde que se está interesado con la formulación de la actualización de pesos con una sola época de entrenamiento [8].

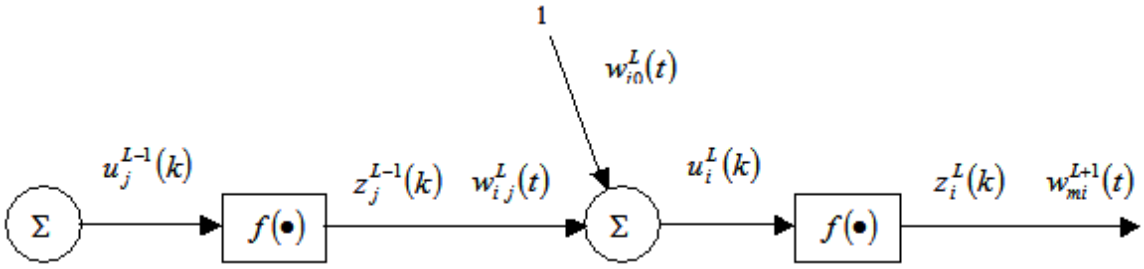


Figura 2.22:Notación utilizada en un modelo de red neuronal PMC multicapa [8]

Para derivar la ecuación de adaptación de pesos, $\partial E / \partial w_{ij}^L$ debe ser computada:

$$\frac{\partial E}{\partial w_{ij}^L} = -2\sum_{k=1}^{K} \frac{\partial E}{\partial u_i^L(k)} \cdot \frac{\partial u_i^L(k)}{\partial w_{ij}^L} = -2\sum_{k=1}^{K} \left[\delta_i^L(k) \cdot \frac{\partial}{\partial w_{ij}^L} \sum_m w_{im}^L z_m^{L-1}(k) \right] = -2\sum_{k=1}^{K} \delta_i^L(k) \cdot z_j^{L-1}(k) \quad (2.39)$$

En la ecuación (2.39), la salida $z_j^{L-1}(k)$ puede ser evaluada aplicando la *k*-ésima muestra de entrenamiento $x(k)$ al PMC con pesos determinados por w_{ij}^L. Sin embargo, el término del error delta $\delta_i^L(k)$ no está disponible actualmente y tiene que ser computado [8].

Se recalca que el error delta estáa definido como $\delta_i^L(k) = \partial E / \partial u_i^L(k)$. La figura (2.23) es ahora usada para ilustrar como computar iterativamente $\delta_i^L(k)$ desde $\delta_m^{L+1}(k)$ y los pesos de la $(L+1)$-ésima capa [8].

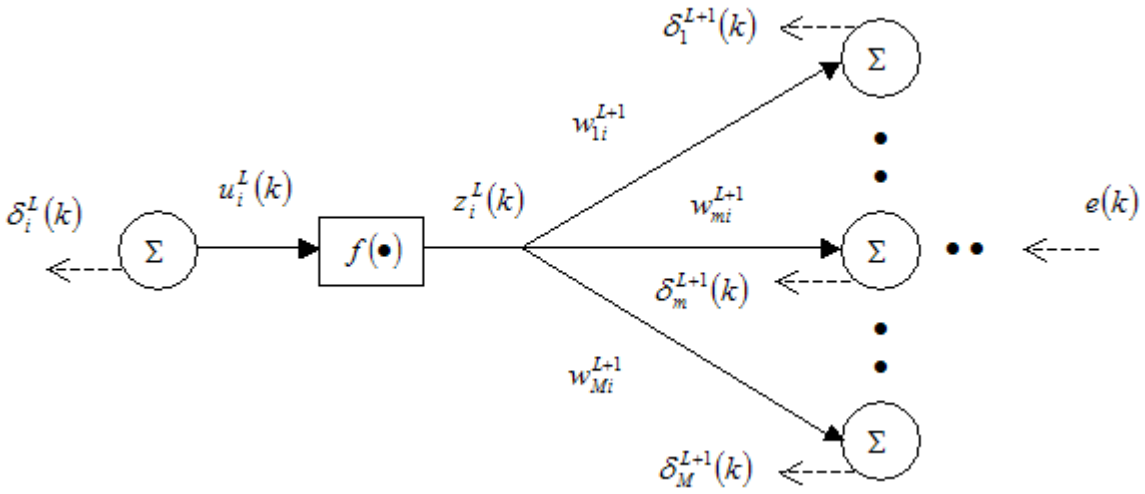


Figura 2.23:Ilustración de cómo la retro propagación del error es computada [8].

Notar que $z_j^L(k)$ es alimentada en todas las M neuronas en la capa $(L+1)$-ésima. De aquí:

$$\delta_i^L(k)=\frac{\partial E}{\partial u_i^L(k)}=\sum_{m=1}^{M}\frac{\partial E}{\partial u_m^{L+1}(k)}\cdot\frac{\partial u_m^{L+1}(k)}{\partial u_i^L(k)}=\sum_{m=1}^{M}\left[\delta_m^{L+1}(k)\cdot\frac{\partial}{\partial u_i^L(k)}\sum_{j=1}^{J}w_{mj}^L f\left(u_j^L(k)\right)\right]=f'\left(u_i^L(k)\right)\cdot\sum_{m=1}^{M}\delta_m^{L+1}(k)\cdot w_{mi}^L \tag{2.40}$$

La ecuación (2.40) es la fórmula de la retro propagación del error que computa el error delta desde la capa de salida hacia la capa de entrada, en una manera de capa por capa.

2.7.3.3 Formulación de la actualización de los pesos con un momentum y ruido

Dando el error delta, los pesos serán actualizados de acuerdo a una formulación modificada de la ecuación (2.37) :

$$w_{ij}^L(t+1)=w_{ij}^L(t)+\eta\cdot\sum_{k=1}^{K}\delta_i^L(k)z_j^{L-1}(k)+\mu\left[w_{ij}^L(t)-w_{ij}^L(t-1)\right]+\varepsilon_{ij}^L(t) \tag{2.41}$$

Del lado derecho de la ecuación (2.41), el segundo término es el gradiente del error cuadrático medio respecto a w_{ij}^L. El tercer término es conocido como el término de momentum. Este provee un mecanismo para ajustar adaptativamente el tamaño del paso. Cuando el vector gradiente en épocas sucesivas apunta a la misma dirección, el tamaño del paso se incrementará (ganando momentum). Cuando los vectores gradiente

sucesivamente forman un patrón de búsqueda en zigzag, la dirección del gradiente efectiva será regulada por este término de momentum para que éste ayude a minimizar el error cuadrático medio [8].

Hay dos parámetros que deben ser escogidos: el factor de aprendizaje, o el tamaño del paso η, y la constante del momentum μ. Ambos parámetros serán escogidos en el intervalo de [0 1]. En la práctica, η frecuentemente asume un valor pequeño, p.e. $0 < \eta < 0.3$, y μ usualmente asume un valor largo p.e. $0.6 < \mu < 0.9$ [8].

El último término en la ecuación (2.41) es un término pequeño de ruido aleatorio que tendrá un pequeño efecto cuando el segundo o tercer término tenga magnitudes grandes. Cuando la búsqueda alcanza un mínimo local o una meseta, la magnitud del vector gradiente correspondiente o el término momentum es apto para disminuir. En esta situación, el término del ruido puede ayudar para sacar al algoritmo de aprendizaje del mínimo local y continuar la búsqueda para la solución óptima global [8].

2.8 Red Feedforward

Esta sección presenta la arquitectura de la red multicapa feedforward que es la más usada con el algoritmo de retro propagación.

El perceptrón multicapa es un modelo de red en el cual las neuronas son configuradas en capas, por lo cual las neuronas de una capa están generalmente todas conectadas con las neuronas de la siguiente capa. Como conexiones existen solamente desde la capa de entrada en la dirección de la capa de salida, está es un red feedforward.

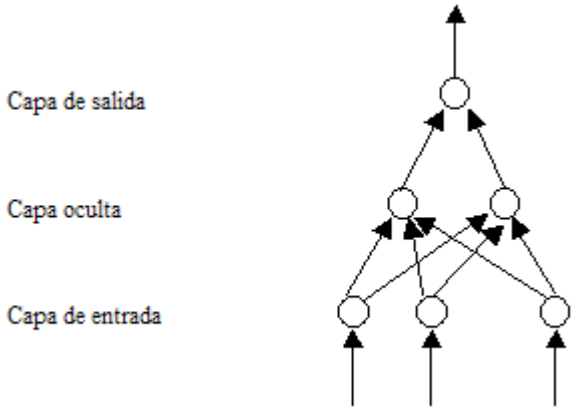


Figura 2.24: Ejemplo de la Topología de una Red Neuronal Feedforward

Las redes multicapa frecuentemente usan la función de transferencia sigmoidea.

Las redes feedforward frecuentemente tienen una o más capas ocultas de neuronas sigmoideas seguidas por una capa de salida de neuronas lineales. Múltiples capas de neuronas con funciones de transferencia no lineales permiten a la red aprender relaciones no lineales y lineales entre los vectores de entrada y salida. Las capas de salida lineales permiten a la red producir valores fuera del rango -1 a +1.

De otra forma, si se quiere forzar las salidas de una red (tal como entre 0 y 1), entonces la capa de salida debe usar una función de transferencia sigmoidea.

Esta red es hábil para procesar patrones de entrada análogos y aprender en modo supervisado, empleando el algoritmo de retro propagación, por lo cual ésta es también referida frecuentemente como la red de retro propagación. Las neuronas son expandidas por un factor umbral y se emplea la función sigmoidea como la función de activación, ya que el algoritmo de aprendizaje requiere la función de activación para ser constante y diferenciable.

La capacidad de ejecución de la red es fuertemente dependiente del número de capas ocultas, ya que éstas son responsables para la capacidad de representación de la red. La siguiente figura muestra la capacidad de partición para diferentes números de capas.

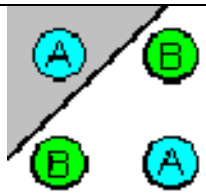
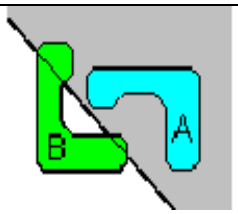
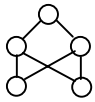
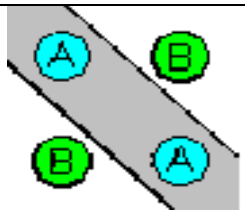
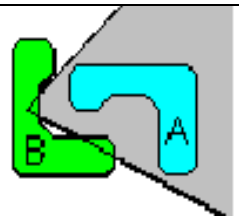
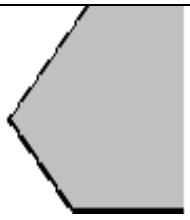
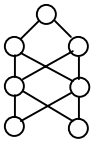
Estructura	Problema de la XOR	Clases con regiones mezcladas	Formas de Regiones más Generales	Regiones de Decisión
2 Capas 				Medio plano limitado por un hiperplano.
3 Capas 				Regiones cerradas o convexas.
4 Capas 				Complejidad arbitraria limitada por el número de neuronas.

Figura 2.25: Muestra la capacidad de partición de un PMC para diferentes números de capas [30]

Como la figura (2.25) lo muestra, una red de 2 capas está solamente habilitada para efectuar separación lineal del espacio de decisión. Esto corresponde a una función de separación lineal. Las redes de 3 capas combinan los resultados de muchas redes de 2 capas y éstas tienen la habilidad para delimitar el espacio de un blanco con una línea continua, el número de bordes el cual corresponde a el número de neuronas ocultas. Esta región es convexa en la forma, sin embargo. Una expansión de la capa intermedia habilita la separación de límites para ser formadas más finamente, pero la restricción de configuraciones convexas no puede ser superada. Finalmente, redes de 4-capas tiene la habilidad de ejecutar cualquier tipo de ejecución requerida de separación, incluyendo configuraciones no-convexas, regiones desconectadas e inclusiones.

Debido a esto, el algoritmo es fácilmente comprensible, el perceptrón multicapa es desarrollado en muchos campos de aplicaciones prácticas. Se señala, sin embargo, que el proceso de entrenamiento puede requerir niveles muy altos de la capacidad de la computadora y podría entonces estar consumiendo mucho tiempo en una computadora convencional. También, la convergencia del algoritmo no está siempre garantizada.

2.9 Algoritmo General de la Red FeedForward

-Se especifica el número de las neuronas de entrada, las ocultas y las de salida

-Se especifica las funciones de transferencia entre capas

-Se preprocesa el conjunto de los datos de entrenamiento para normalizarlos

-Se entrena la red feedforward con el error de retro propagación del PMC para encontrar los pesos. El entrenamiento necesita las entradas y salidas.

-Se prueba la red, con los datos de entrenamiento y la salida conocida.

-Se prueba la red con datos nuevos para ver su funcionamiento

CAPÍTULO 3
Desarrollo del Sistema

En el sistema de la figura (3.1) es donde se muestra el cumplimiento de los objetivos planteados.

- Corregir la distorsión producida por el lente gran angular
- Mejorar el muestreo y la clasificación de colores
- Utilizar más de un dispositivo de captura en el sistema de visión

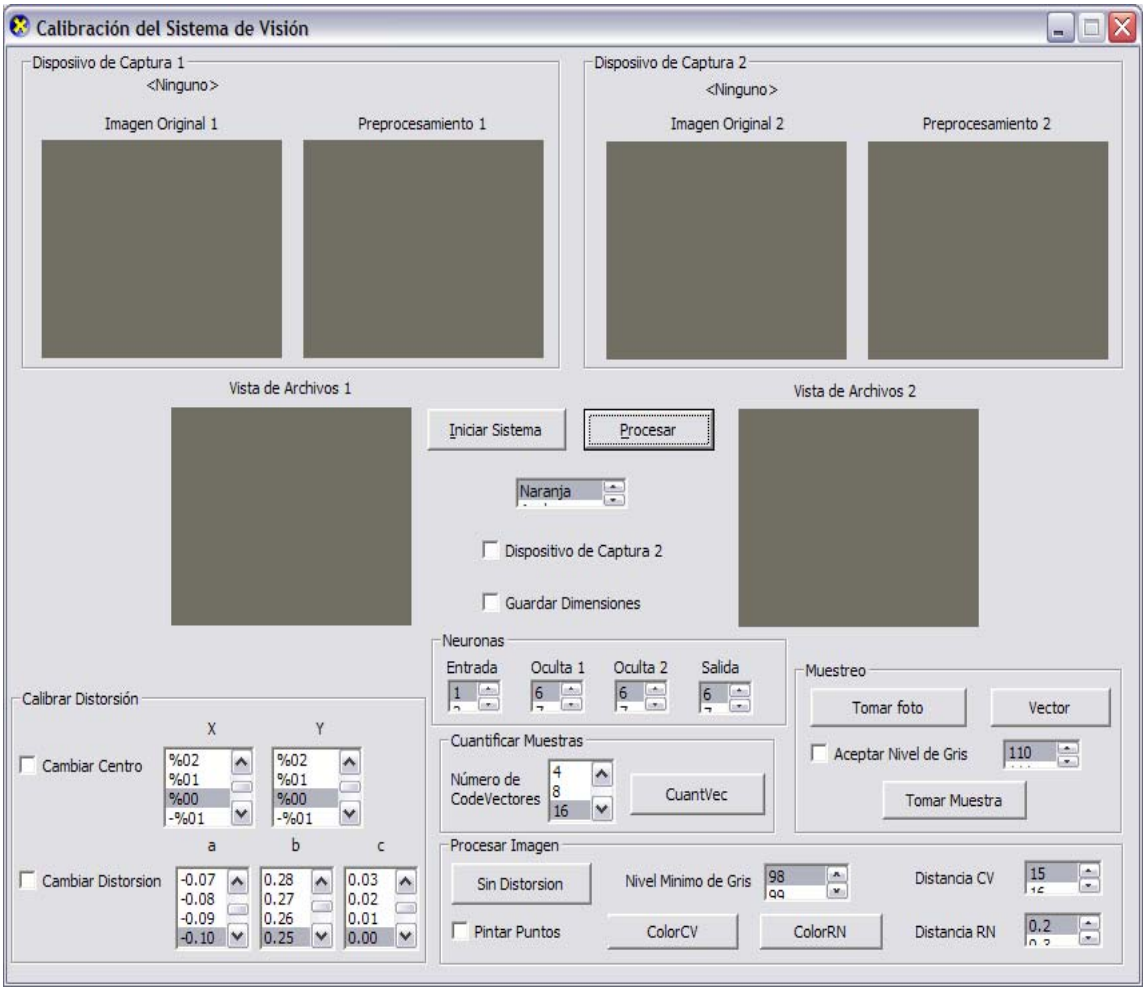


Figura 3.1: Sistema de Calibración del Sistema de Visión

3.1 Reconocimiento de Patrones

Para mejorar la recolección de muestras, se plantearon los siguientes métodos: Árbol cuaternario, Detección de Bordes, VCC y Autómata de intersección de líneas. A través de métodos se desea reconocer la información de un patrón de rejilla.

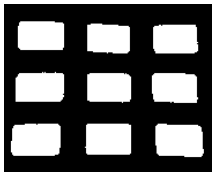


Figura 3.2:Patrón de rejilla

Por cada color a clasificar, se formará el patrón de rejilla para tener muestras de las diferentes partes del campo de juego, ya que en todo el campo de juego varía la intensidad de la luz y por lo tanto de los colores.

3.1.1 Árbol Cuaternario

El árbol cuaternario se utiliza en un algoritmo de fusión y división. Esto consiste en dividir inicialmente la imagen en un conjunto de regiones arbitrarias disjuntas para después fusionar y/o dividir estas regiones de acuerdo a alguna condición lógica y a la representación final que se quiere obtener

Sea R la región correspondiente a la imagen completa y P un predicado lógico. Entonces se divide R de forma sucesiva en regiones cada vez más pequeñas de modo que, para cualquier región R_i, $P(R_i)$ sea verdadero y donde $P(R_i)$ sea falso se divide R_i en cuadrantes. Esta técnica de división se representa por medio de los árboles cuaternarios. En la siguiente figura se muestra el árbol cuaternario de una imagen, la raíz del árbol es la imagen completa.

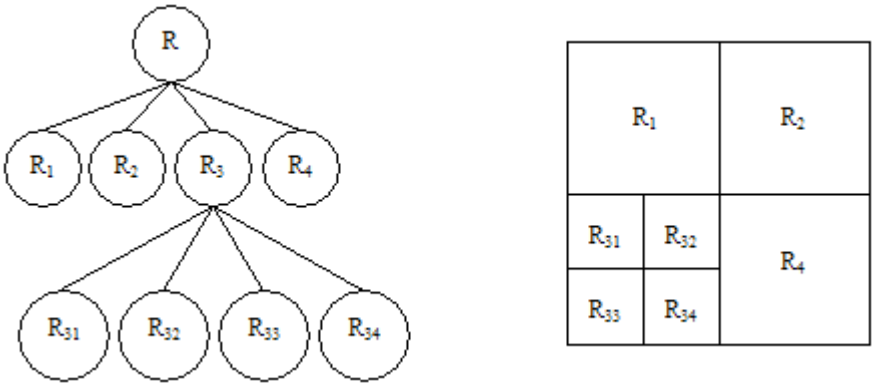


Figura 3.3:Árbol cuaternario y la imagen correspondiente dividida

Si solo se usa la división, la partición final estará formada por regiones adyacentes con propiedades idénticas. Si al dividir la imagen se tiene regiones adyacentes, se usa la fusión de regiones adyacentes con propiedades idénticas, para ir segmentando la imagen. Cuando ya no se pueda hacer más divisiones ni fusiones, se habrá terminado el proceso.

3.1.2 Detección de Bordes

Los bordes caracterizan los límites de un objeto, por lo que los bordes presentan una transición de claro a oscuro o viceversa. Por lo tanto, se detectan calculando el gradiente de la imagen en dos direcciones ortogonales. La siguiente figura muestra dos operadores gradiente.

	Sobel	**Roberts**
Horizontal	$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$	$\begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$
Vertical	$\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$	$\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$

Figura 3.4:Operadores gradiente para la detección de bordes

Los operadores gradiente horizontales, detectan bordes horizontales, y los otros, los verticales. Obteniendo así dos imágenes de gradiente I_{gx}, I_{gy}. Para determinar si un píxel pertenece al contorno, se define un umbral, por lo cual la imagen de contornos *I(x,y)* se forma a partir de:

$$I(x,y) = \begin{cases} 1 & si \quad |I_{gx}(x,y)| + |I_{gy}(x,y)| > umbral \\ 0 & en\ otro\ caso \end{cases}$$

3.1.3 VCC (Vertex Chain Code)

El VCC es un código de cadena, el cual está basado en el número de vértices que están en contacto en el contorno de la figura. Algunas de las características del VCC es que es invariante bajo traslación y rotación, opcionalmente puede ser invariante bajo un punto de inicio y a la transformación espejo. También usando el VCC es posible obtener relaciones entre el límite del contorno y el interior de la figura. La siguiente figura muestra un ejemplo del VCC.

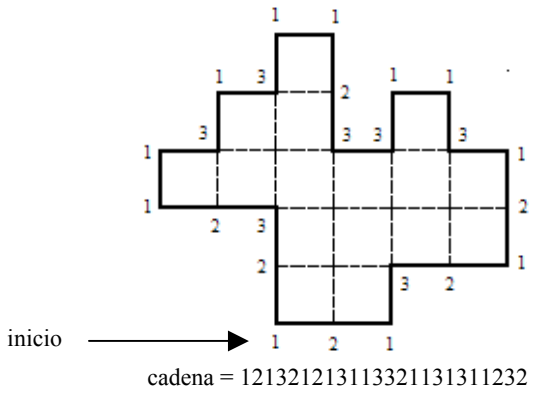


Figura 3.5:En el contorno de la figura está escrita la cadena del VCC [9]

3.1.4 Autómata de Intersección de Líneas

Esta técnica se desarrolló con las cualidades de las técnica antes mencionadas. El VCC da la coordenada inicial del borde de la figura y una cadena descriptiva de dicha figura, además de poder conocer con esta cadena la exactitud del área de dicha figura. La detección de bordes encuentra todos los contornos en la imagen. El árbol cuaternario da la idea de la división y fusión para segmentar la imagen.

Para desarrollar el autómata se inicia por detectar bordes a través de una "línea", al encontrar la línea se guarda la coordenada hasta que no encuentre un borde, al terminar de revisar la imagen se tiene la imagen dividida por las coordenadas de intersección y se procede a unir los datos de estas divisiones por medio de un nivel mínimo de gris.

En la figura (3.6) se muestran los controles para utilizar el autómata de intersección de líneas. El procedimiento es el siguiente, se usa "Tomar foto" que detiene la toma del buffer del dispositivo de captura. Si la imagen o foto de este buffer tiene las características deseadas, entonces se guarda el buffer ("Vector"). El buffer está en forma de vector, por lo que al guardarlo se cambia a matriz donde cada coordenada contiene un píxel en la forma BGR y HSI. Por lo tanto se guardan 2 archivos del buffer y en su encabezado se tienen las dimensiones del buffer en forma de matriz que corresponden a las dimensiones del dispositivo de captura.

Figura 3.6:Controles para la toma de muestras

Para tomar los datos deseados de la imagen guardada, se usa "Tomar Muestra" para abrir los archivos guardados y mostrar el contenido del archivo RGB en "Vista de Archivo 1" o "Vista de Archivo 2". La imagen mostrada en "Vista de Archivos" está binarizada con el operador umbral por medio de un nivel de gris. Si esta imagen es lo que se quiere ver para tomar las muestras, entonces hay que "Aceptar Nivel de Gris" para guardar las muestras con el formato del espacio de color HSI.

Figura 3.7: Ventanas para desplegar los archivos guardados

3.2 Calibración del Sistema de Visión

El siguiente diagrama muestra la parte de los pasos fundamentales del Procesamiento Digital de Imágenes donde se realizaron las mejoras para el Sistema de Visión.

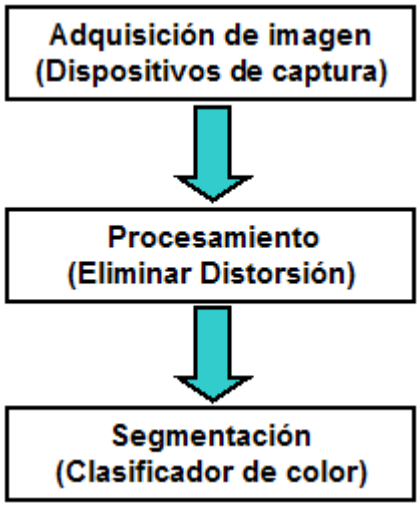


Figura 3.8:Diagrama donde se muestran las mejoras

3.2.1 Utilizar más de un dispositivo de captura (cámaras)

El sistema de Calibración del Sistema de Visión, basa la adquisición de la imagen (captura del buffer) en DirectShow [5], que es una arquitectura de Microsoft para audio y video, proveyendo captura y reproducción de alta calidad para flujos multimedia.

La siguiente gráfica muestra los pasos utilizados para desplegar los datos que provienen de los dispositivos de captura.

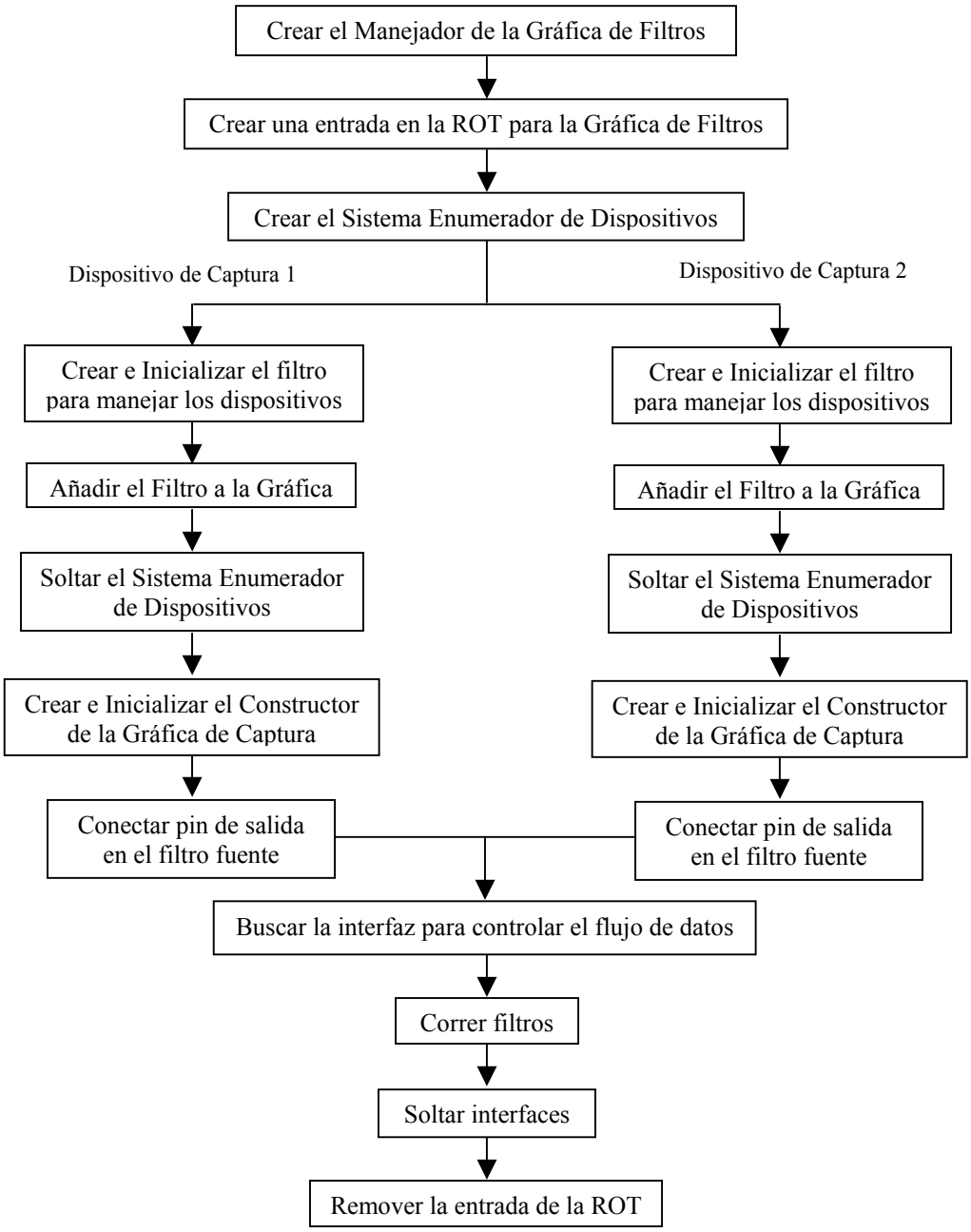


Figura 3.9:Diagrama para desplegar datos de los dispositivos de captura

Utilizando este diagrama, se observan los datos del dispositivo de captura en "Imagen Original 1" y/o "Imagen Original 2". Para procesar el buffer (datos) del dispositivo de captura se deben manejar eventos, ya que la aplicación de DirectShow necesita una manera para encontrar una salida cuando los eventos están esperando en la cola. El Manejador de la Gráfica de Filtros [Apéndice A] da acceso a la técnica de Window notification [Apéndice A] para manejar los eventos. Los datos procesados son desplegados en "Preprocesamiento 1" y/o "Preprocesamiento 2"

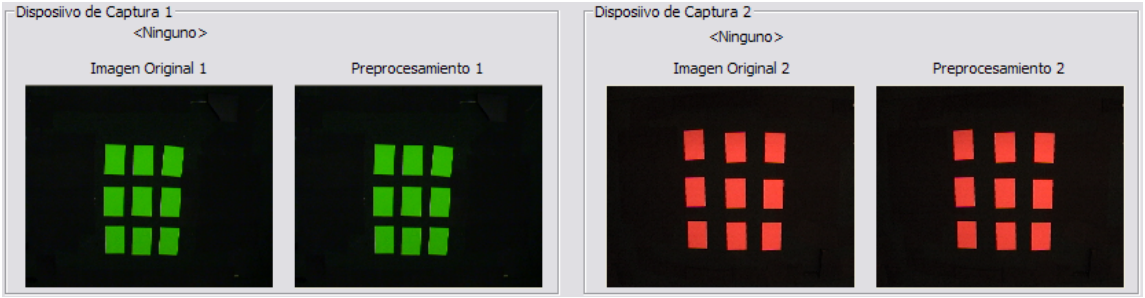


Figura 3.10:Ventanas donde se despliegan las imágenes capturadas y las imágenes procesadas.

3.2.2 Corrección de la distorsión producida por el lente gran angular

En este paso se calibra manualmente la distorsión del lente, se hace mediante un patrón que contenga líneas rectas para ir observando la corrección al cambiar las variables (a, b, c, x, y).

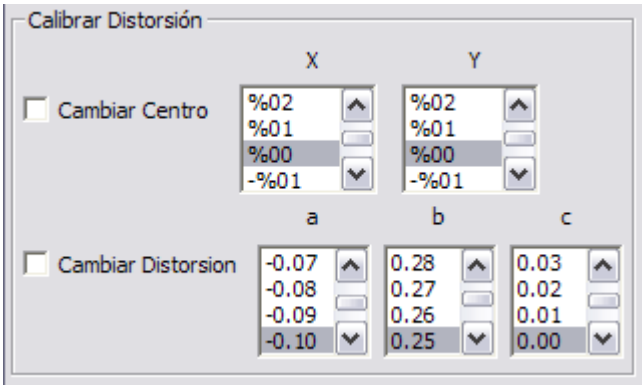


Figura 3.11:Controles para eliminar la distorsión de la imagen.

La fórmula que se utiliza para quitar la distorsión es la siguiente:

$$r_{src} = \left(a * r^{3}{}_{dest} + b * r^{2}{}_{dest} + c * r_{dest} + d\right) * r_{dest} \qquad (3.1)$$

Las variables r_{src} y r_{dest} describen el radio que tiene un rango de *[0,1]*. Una imagen es representada en un espacio bidimensional y para localizar algún píxel (dato) de la imagen se hace por medio de coordenadas cartesianas. Esto implica que para poder eliminar la distorsión se debe de convertir las coordenadas cartesianas en coordenadas polares.

Al convertir las coordenadas cartesianas en coordenadas polares, se utiliza la función *acos( )* para obtener el ángulo de la coordenada polar de la siguiente manera:

$$\theta = a\cos\left(\left(x - centrox\right)/radio(x,y)\right) \tag{3.2}$$

Al usar la función *acos( )*, se hace con *x-centrox*, ya que $x=r\ cos\theta$. Esto hace que θ vaya de $[0,\pi]$, por lo que se tiene lo siguiente:

$$a\cos((x-centrox)/radio(x,y)) = \begin{cases} 0 > \theta > \pi/2, & si\ x < centrox \\ \theta = \pi/2, & si\ x = centrox \\ \pi/2 > \theta > \pi, & si\ x > centrox \end{cases}$$

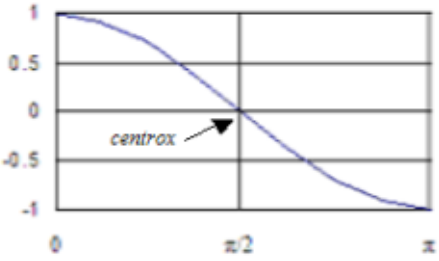


*Figura 3.12:Valor de **θ**, a partir del centrox*

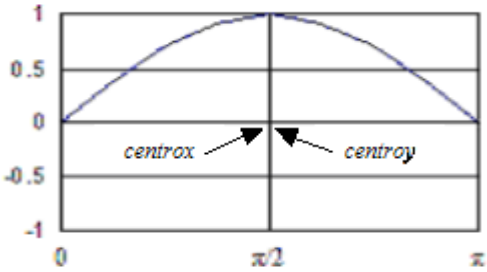


Figura 3.13:Función seno de [0, π], a partir del centrox

Por lo anterior, si uno quiere variar la posición del *centrox* no hay ningún problema, ya que los ángulos están bien definidos. Pero al variar *centroy* existe la dificultad de que θ se tomó respecto a *centrox*, por lo que al evaluar $y=r\ sen\theta$ se tiene que los valores de $sen\theta$ en el rango de [0, π] son todos positivos y con ello dos píxeles ocuparán la misma *"y"*, por lo cual hay que tener una condición cuando θ está en el rango (π/2, π]. Ahora si movemos *centroy* del centro de la imagen, además del problema ya mencionado, se agrega el desplazamiento de este movimiento. A continuación se muestra un algoritmo para asignar correctamente la coordenada *"y"*, la sintaxis es similar a la de C.

```
1  difpmy=centroyimag-centroy; //diferencia para corregir el desplazamiento
2  //ciclos para corregir la distorsión de la imagen
3  for(i=0; i<y; i++)
4  {
5    for(j=0; j<x; j++)
6    {
7    //Se toma el ángulo para la coordenada polar, respecto a "x"
8    . . . . .
9     //se aplica la fórmula de la distorsión para encontrar la nueva
10    //distancia (nuevadist), que es el radio del píxel sin distorsión
11   . . . . .
12   //se desnormaliza la distancia
13   . . . . .
14   //centrox es la coordenada donde la imagen no está distorsionada
15   coordx=nuevadist*cos(angulox)+centrox;
16   //centroy es la coordenada donde la imagen no está distorsionada
17   if(i< centroy)//si la "y" está en el rango (π/2, π]
18    {
19       coordy=nuevadist*sin(angulox)+centroy;
20       coordy=y-coordy;
21       //corrección del desplazamiento
22       if(difpmy!=0) coordy = coordy - (difpmy*2);
23    }
24   else // si la "y" esta en el rango [0,π/2]
25       coordy=nuevadist*sin(angulox)+centroy-1;
26   }
27 }
```

En la figura (3.11) el centro de la distorsión en el porcentaje (0,0), es el centro de la imagen, por lo que si se quiere cambiar el centro de la distorsión, se tiene que:

- si X se mueve en un porcentaje positivo, el centro se mueve hacia la derecha
- si X se mueve en un porcentaje negativo, el centro se mueve hacia la izquierda
- si Y se mueve en un porcentaje positivo, el centro se mueve hacia arriba
- si Y se mueve en un porcentaje negativo, el centro se mueve hacia abajo.

La distorsión producida por los lentes gran angular del Laboratorio de Bio-Robótica, presentan la distorsión de barril. Al estar calibrando el sistema solo fue necesario eliminar la distorsión con el parámetro "*b*". Por lo que la ecuación (3.1) queda de la siguiente manera:

$$r_{src} = b * r^{3}_{dest} + d * r_{dest} \tag{3.3}$$

Si la variable *b* es positiva, se corrige la distorsión de barril y si es negativa, se corrige la distorsión de cojín.

A continuación se muestra un ejemplo de como corrige la distorsión de barril con el parámetro b=0.1.

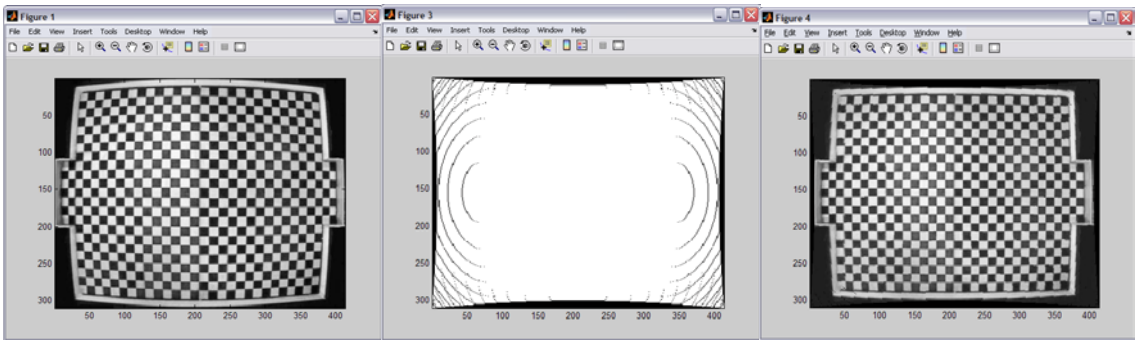


Figura 3.14:Ejemplo de la corrección de barril con el parámetro b=0.1

Para observar la corrección de la distorsión, se utiliza el botón "Sin Distorsión" de la figura (3.1). Este proceso crea una matriz de coordenadas las cuales son las coordenadas ya corregidas. Por ejemplo en la matriz, la coordenada (20,20) contiene la coordenada corregida (23,20). De esta forma se reasignan todos los píxeles en una matriz y después se transforma a vector, ya que la aplicación despliega la imagen del buffer en forma de vector.

La casilla "Pintar Puntos" de la figura (3.1), es útil para ver qué tanto varia la posición de los puntos al corregir la distorsión. El proceso es que se dan 2 puntos y se pintan en el buffer de azul, al usar el botón "Sin Distorsión" se pintan los puntos azules y unos puntos rojos, los puntos rojos son los puntos corregidos (sin distorsión).

3.2.3 Reconocimiento de colores por el Cuantificador Vectorial o la Red Neuronal feed forward.

El cuantificador vectorial es un buen descriptor de espacios n-dimensiónales, de hecho se puede usar como un compresor de información. La RNA tiene la cualidad de poder encontrar posibles soluciones a problemas que, solucionados con modelos matemáticos tradicionales no tienen una solución satisfactoria.

3.2.3.1 Cuantificador Vectorial

El tipo de cuantificador vectorial utilizado es el Método de Splitting [4], este método garantiza que el número de codevectores de salida es par y que ninguna partición a la cual representan los codevectores esté vacía. Al ejecutar el cuantificador, cada que realiza la partición de los codevectores se les asigna vectores que estén mas cerca del codevector y esto lo realiza a través de la distancia euclidiana. Para estabilizar los codevectores se aplica el Algoritmo de Lloyd, en esta etapa es donde se tiene el

cuidado de que ninguna partición representada por los codevectores esté vacía. Cuando una o más regiones están vacías, después de varias iteraciones del Algoritmo de Lloyd y la técnica de la división del codevector con más vectores pertenecientes a su espacio, no cambia la condición de vacío en las regiones, entonces se procede a escoger aleatoreamente codevectores y se les suma o se les resta aleatoreamente un factor de ruido para que algún vector que esté en el límite de la región del codevector pase a una región vacía. Este proceso se realiza hasta que todas las particiones no estén vacías.

Para utilizar el cuantificador vectorial, se deben cargar los archivos que contiene el codebook de cada color. Para ello se marca el número de codevectores del codebook y el color a utilizar.

Figura 3.15:Controles para elegir el tamaño y ejecutar el cuantificador vectorial

El número de codevectores que mejor rendimiento tienen en cuanto a clasificación y velocidad de procesamiento son de tamaño 8 y de 16 codevectores por cada color a clasificar.

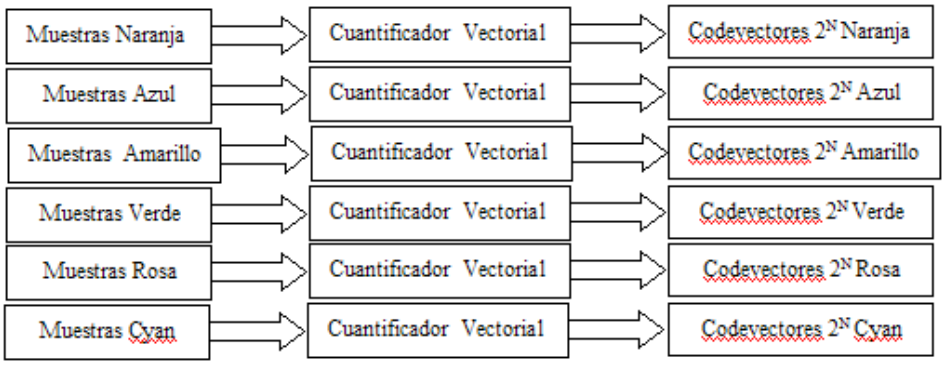


Figura 3.16:Esquema de entrenamiento para el cuantificador vectorial

3.2.3.2 Red Neuronal feed forward

La red neuronal feed forward, se utiliza por su capacidad de procesar patrones de entrada análogos y por aprender en modo supervisado, empleando el algoritmo de retro-propagación. Además de que las soluciones están relacionadas con el número de capas ocultas, ver figura (2.25), y por ello es más fácil de comprender qué diseño de red se necesita para encontrar una solución.

Para utilizar la red neuronal feed forward, se define el número de neuronas de la capa de entrada, el número de neuronas da la capa oculta y el número de neuronas de la capa de salida.

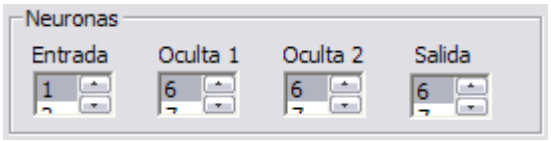


Figura 3.17:Controles para definir las neuronas de las capas

Por ejemplo, si en el entrenamiento se definen 3 neuronas de entrada, 2 capas ocultas de 6 neuronas cada una y 6 neuronas de salida se crea una red como la figura (3.18). A continuación se muestra las estructuras de la red que más se usan para el sistema de visión.

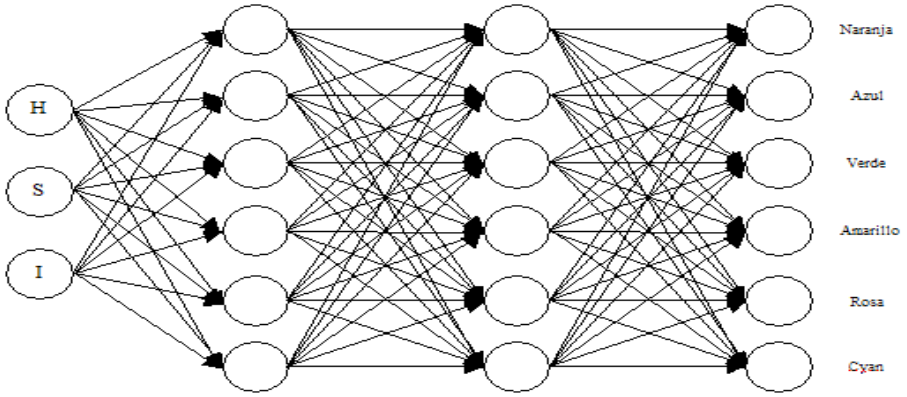


Figura 3.18:Esquema de una RNA feed forward con 3 neuronas de entrada, 2 capas ocultas de 6 neuronas cada una y 6 neuronas de salida

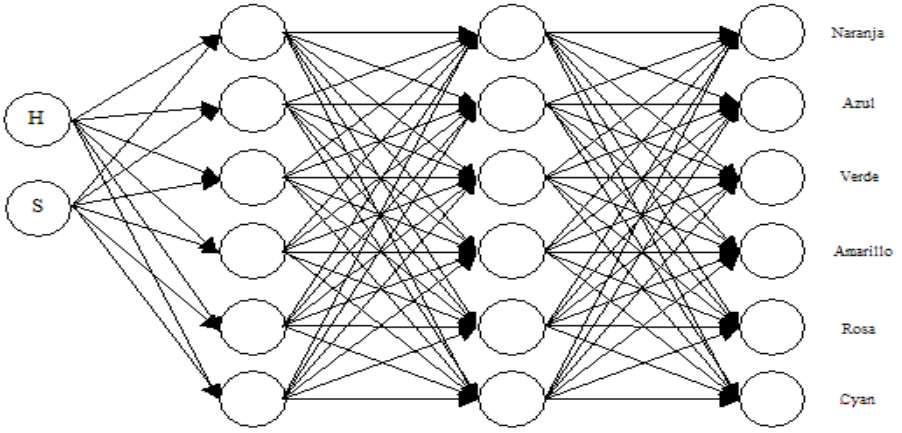


Figura 3.19:Esquema de una RNA feed forward con 2 neuronas de entrada, 2 capas ocultas de 6 neuronas cada una y 6 neuronas de salida

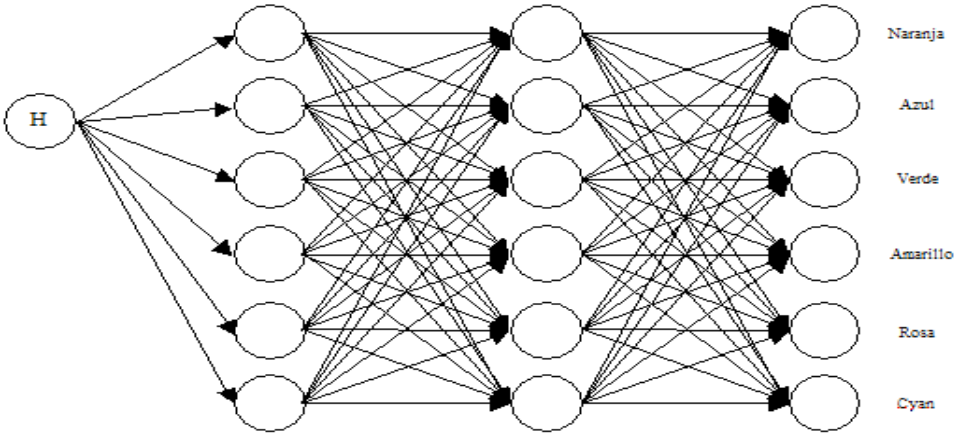


Figura 3.19:Esquema de una RNA feed forward con 1 neurona de entrada, 2 capas ocultas de 6 neuronas cada una y 6 neuronas de salida

Al estar entrenando la red neuronal con diferentes parámetros, como el número de neuronas de entrada, el número de neuronas en la capa oculta, el factor de aprendizaje, el momentum, el número de épocas y las funciones de transferencia, se observó que al entrenar la red con momentum desciende rápidamente el error cuadrático medio. El número de épocas tiene una relación fuerte con el número de neuronas de entrada y con la cantidad de datos de entrenamiento para la red. El factor de aprendizaje después de varios entrenamientos varia entre 0.15 y 0.17, valores más pequeños necesitan una mayor cantidad de épocas para minimizar el error y no por ello se llega a un error mínimo aceptable, con valores más grandes se pueden necesitar menos épocas pero no se llegó a un error mínimo aceptable. El número de neuronas de entrada, era en un principio de tres neuronas (HSI), pero al observar la salida de la fase de pruebas de la red después de haberla entrenado, se vio que la I tenía un 0 en su clasificación, esto indica que no ocupa la I para clasificar los colores. Por lo que el diseño está entre 1 y 2 neuronas de entrada. El diseño de 2 neuronas de entrada necesita un trato especial ya que a veces clasifica por la saturación, por lo que se usa la red neuronal con 1 neurona de entrada, esta puede ser solo H o HS siendo H parte alta del dato de entrada y S la parte baja. La función de red que se usa para determinar como las entradas de la red $\{y_j; 1 \le j \le N\}$ son combinadas dentro de la neurona es la combinación lineal de pesos:

$$u = \sum_{j=1}^{N} w_j y_j + \theta \tag{3.4}$$

donde $\{w_j; 1 \le j \le N\}$ son los pesos sinápticos y θ es el umbral (o bias). La función de transferencia usada para las dos capas es la sigmoidea (logsig):

$$f(u) = \frac{1}{1+e^{-u}} \tag{3.5}$$

esta función toma los valores de entrada, los cuales pueden oscilar entre más y menos infinito, y restringe la salida a los valores entre [0,1], además de que la función sigmoidea es diferenciable y esto es de gran utilidad para la retro-propagación.

La red neuronal se entrenó de la siguiente forma: Con todas las muestras tomadas y con los codevectores del cuantificador vectorial.

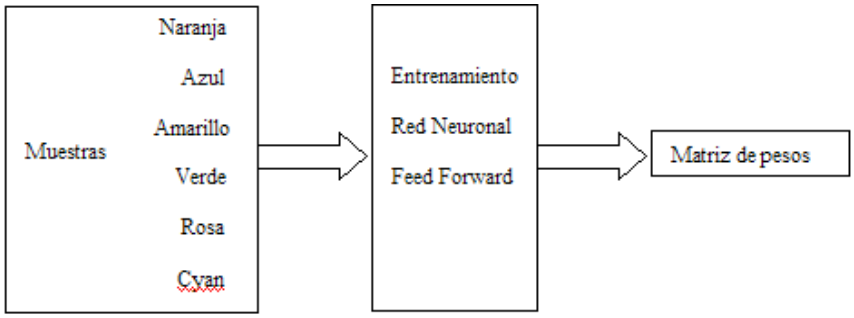


Figura 3.20:Esquema de entrenamiento de la RNA feed forward con todas las muestras

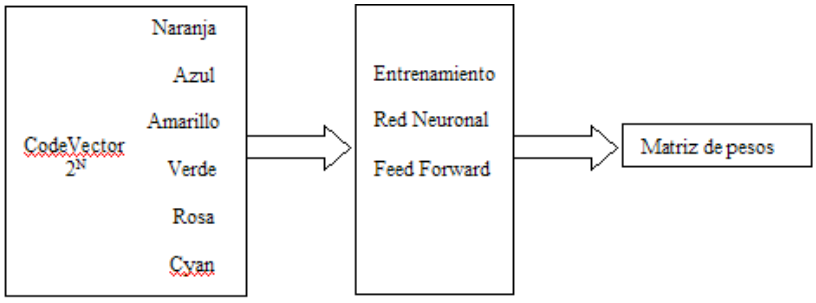


Figura 3.21:Esquema de entrenamiento de la RNA feed forward con los codevectores

3.2.3.3 Segmentación de la imagen a través de la clasificación del color

Para describir cómo el sistema hace su segmentación a través de la clasificación de color se hará mención de los controles de la figura (3.22).

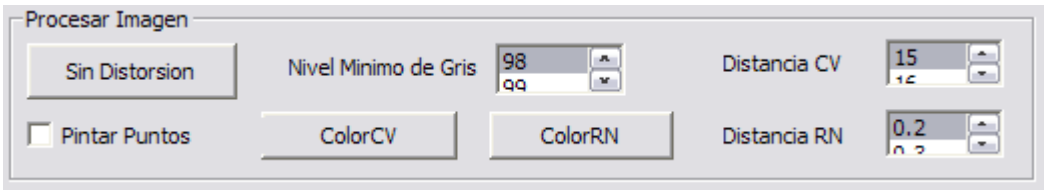


Figura 3.22:Controles del sistema de calibración para observar la segmentación de color

Se utiliza el botón “ColorCV” el cual procesa el buffer de la ventana de “Preprocesamiento”. El procesamiento se realiza revisando cada píxel de la imagen para ver a qué color pertenece, esta pertenencia se obtiene a través de los codevectores del cuantificador vectorial. Para saber si un píxel (vector de entrada *x*) pertenece a algún color, se calcula con la distancia euclidiana del píxel a la suma de las distancias euclidiana del codebook de cada color.

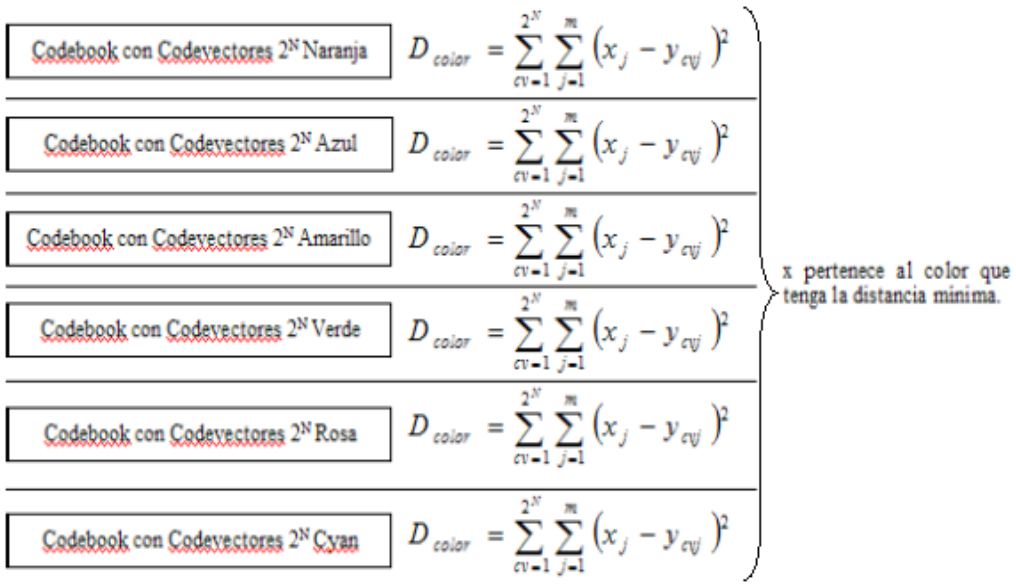


Figura 3.23:Método de clasificación a través del Cuantificador Vectorial

Como la distancia euclidiana entre el píxel y el codebook de cada color es la medida que clasifica, para eliminar ruido en la clasificación se tiene una lista de valores de la distancia euclidiana mínima, para abrir o cerrar el marco de pertenencia.

Para poder ver la clasificación de la red neuronal, se utiliza el botón “ColorRN”. El procesamiento es revisando cada píxel (neurona(s) de entrada de la red) de la imagen para ver la pertenencia del color, Esta pertenencia es dada por la red neuronal, ya que al clasificar la red neuronal regresa un vector que contiene un valor entre 0 y 1, el tamaño del vector es del número de colores a clasificar (neuronas de salida de la red). De los valores regresados en el vector, se toma el valor más grande y ese será el color ganador.

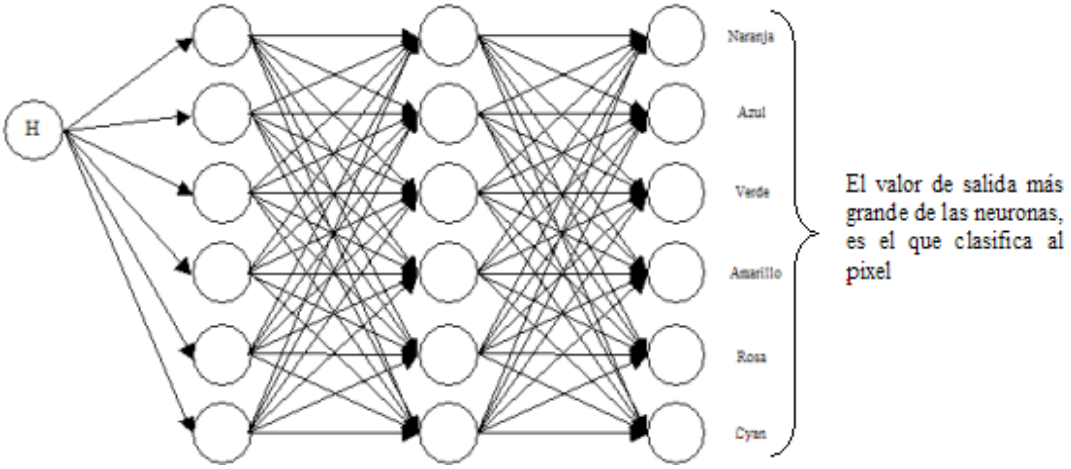


Figura 3.24:Método de clasificación a través de la RNA feed forward

Para poder tener un marco de pertenencia, se tiene una lista de valores que van entre 0 y 1, se escoge de la lista "DistanciaRN" la distancia mínima de pertenencia.

Estos procedimientos al procesar el buffer, para clasificar los colores son pesados para el procesador, ya que se revisa píxel por píxel y los clasificadores trabajan en el espacio de color HSI, mientras que el buffer está en el espacio de color RGB. Por lo que implica transformar cada píxel de RGB a HSI.

Al observar que el muestreo utiliza un nivel de gris para excluir los píxeles que no tienen un color para clasificar, entonces no es necesario transformar toda la imagen RGB a HS ya que si el píxel no cumple con un "Nivel Mínimo de Gris" sólo se abrá transformado de RGB a I. Esto implica que sólo se clasificarán los píxeles que cumplan con el nivel de gris. Con esto se acelera bastante el procesamiento ya que el porcentaje de píxeles a clasificar está entre un 15% y 20%

3.3 Sistema de Visión y Procesamiento

Este sistema ha sido desarrollado en el laboratorio de Bio-Robótica, de la Facultad de Ingeniería para el equipo de robots de la categoría de fútbol "small size", por lo que la descripción del sistema esta en [1], algunas de las modificaciones hechas a este sistema ya están definidas en este capítulo, por lo que se detallarán los controles que no se mencionaron en ninguna de las descripciones ya mencionadas.

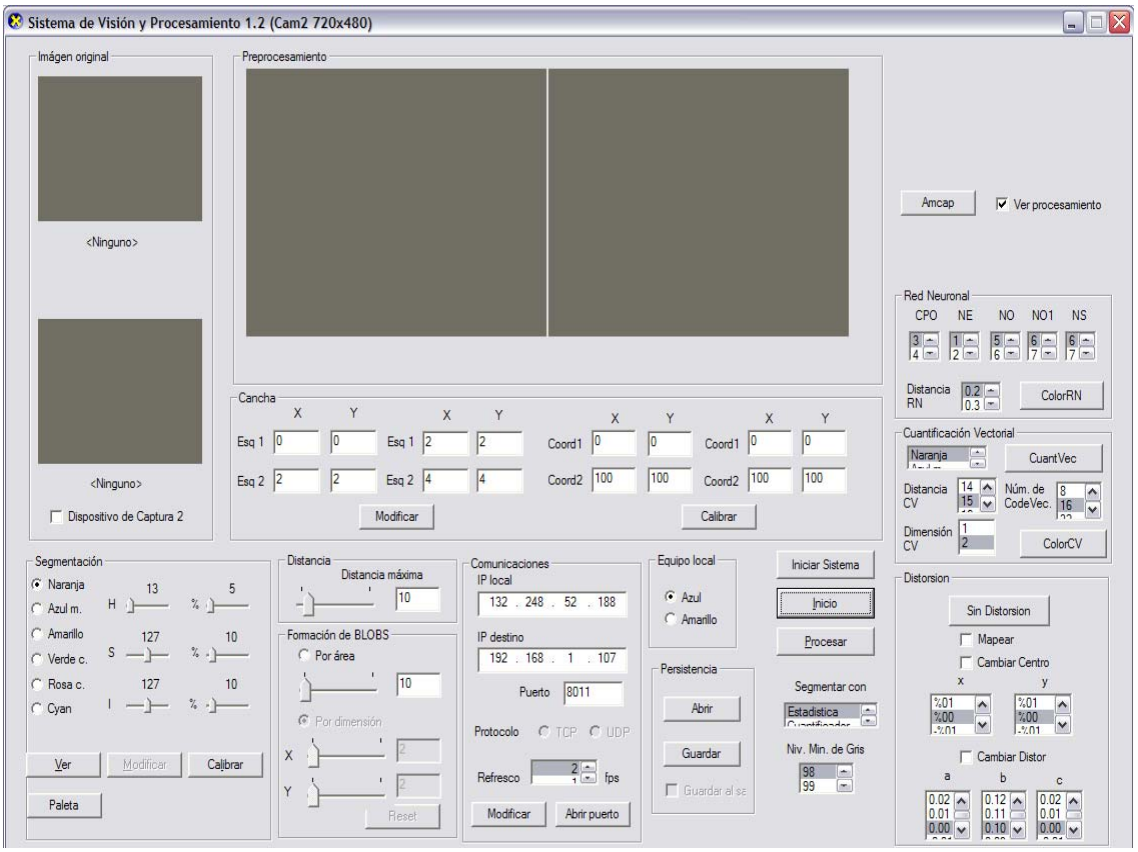


Figura 3.25:Sistema de Visión y Procesamiento

En la etapa de segmentación ahora se tienen 3 técnicas. La primer técnica es la "Estadística", la segunda es la "Cuantificación Vectorial" y la tercera es la "Red Neuronal".

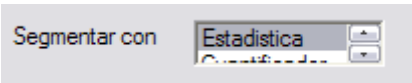


Figura 3.26:Control para elegir el tipo de clasificador

La técnica de la "Estadística" es la primer técnica desarrollada para el Sistema de Visión y utiliza las tres componentes HSI para el reconocimiento de color. La técnica de la "Cuantificación Vectorial" utiliza de dos componentes (HS) a una componente (H) dependiendo de la efectividad al clasificar. La tercer técnica es la "Red Neuronal" la cual también usa de dos componentes (HS) a una componente (H) de acuerdo a su efectividad al clasificar.

En esta etapa es donde la velocidad de procesamiento mejora significativamente, ya que en la técnica de la "Estadística" se necesita transformar todos los píxeles de la imagen de RGB a HSI para poder dar la clasificación de cada píxel; esta transformación

es muy costosa para el procesador debido a que la H necesita de la raíz cuadrada, la suma, la resta, la división, elevar al cuadrado y el coseno inverso. En las otras dos técnicas de segmentación la componente de la Intensidad es la primera en segmentar los píxeles, dejando entre un 15% y 20% de píxeles para segmentar con cualquiera de las dos técnicas, por lo que sólo estos píxeles serán convertidos de RGB a HSI.

Al elegir la "Cuantificación Vectorial" o la "Red Neuronal", sólo se abren archivos que ya contienen los parámetros para la segmentación de cualquiera de estas dos técnicas. Estos archivos fueron creados en el programa de Calibración del Sistema de Visión [Apéndice B].

La casilla de "Mapear", figura (3.25), se utiliza para abrir un archivo que contiene las coordenadas de cada píxel, estas coordenadas ya no tienen la distorsión. Esto no tiene un costo en el procesamiento, ya que sólo se tiene que mapear una coordenada por la posición de la pelota, tres coordenadas para dar la posición de cada robot de nuestro equipo y una coordenada por cada robot del equipo contrario.

La razón de usar una matriz de coordenadas, es porque al corregir la distorsión, se corrige la posición de cada punto en la imagen, por lo que se guarda esta corrección para no estar aplicando la fórmula de corrección a cada coordenada que se necesite corregir cuando se esta realizando el procesamiento.

CAPÍTULO 4

Pruebas y Resultados

En este capítulo se muestran las pruebas y resultados del Sistema de Visión y Procesamiento, donde se observarán las mejoras realizadas a dicho sistema.

Para realizar las pruebas se mencionan a continuación los tres tipos de objetos que se tiene que clasificar en un juego de fútbol: la pelota, los jugadores propios y los jugadores contrarios.

Por lo anterior las pruebas realizadas al sistema se dividen en cuatro, ya que son para el reconocimiento de los objetos en escena.

- Localización de la pelota
- Localización de los robots propios
- Localización de los robots contrarios
- Velocidad de procesamiento

4.1 Pruebas

Las siguientes pruebas tienen el objetivo de conocer el porcentaje de éxito y por lo tanto de error que el sistema tiene al reconocer todos los objetos en la escena. Los errores son influenciados por las características de los dispositivos de captura, por las lentes utilizadas para estos dispositivos, por los algoritmos utilizados en el procesamiento de la imagen, así como por eventos fuera de control como la iluminación.

Las mejoras al sistema desarrollado, minimizan los errores antes mencionados por medio de una mejor segmentación con los diferentes clasificadores desarrollados, por un mejor muestreo y por la corrección de la distorsión del lente.

Las pruebas se harán a través de dos cámaras digitales, con las cuales se mostrará el comportamiento del sistema. El modelo de las cámaras es el siguiente:

1. Sony DCR-TRV380 y
2. Sony DCR-HC46 E33

Para los resultados de las pruebas de la localización de los diversos objetos, se utilizará la medida real del objeto sobre el campo de juego y la medida dada por el Sistema de Visión y Procesamiento.

Estas pruebas son de suma importancia, ya que con una buena identificación de los objetos, una buena estimación de las coordenadas de cada uno de los objetos y un

procesamiento eficiente, se desarrollarán mejor las estrategias del sistema de Inteligencia Artificial.

En las pruebas de localización, se tienen dos tipos de resultados: los estáticos y los dinámicos. Los resultados estáticos servirán para medir la precisión de las coordenadas. Los resultados dinámicos servirán para medir la cantidad de paquetes que recibe el sistema de Inteligencia Artificial.

Los pasos que se realizarán para las pruebas estáticas son los siguientes:

1. El objeto se coloca en escena.
2. Se anota la posición real del objeto en la escena. En el caso de los robots propios también se anotará la orientación el robot.
3. Se anota la información recibida por el Sistema de Inteligencia Artificial.
4. Se compararán los datos del paso 2 con los datos del paso 3.

Los pasos que se realizarán para las pruebas dinámicas son los siguientes:

1. Se coloca el objeto en escena.
2. Los objetos se moverán en forma aleatoria dentro de la escena.
3. Se anota la información recibida por el Sistema de Inteligencia Artificial y se cuentan los paquetes recibidos. Del total de estos paquetes se obtendrá el número de veces que el Sistema de Visión y Procesamiento perdió al objeto para calcular el porcentaje de pérdida de los diferentes objetos.

4.2 Localización de la pelota

La pelota es uno de los objetos en escena, y a partir del cual todo el juego se desarrolla, por lo que es importante su correcta identificación. Sin embargo presenta la particularidad de ser el objeto más difícil de encontrar debido a que es un objeto esférico que refleja la luz en todas direcciones, a diferencia de los parches de identificación de los robots. Además su tamaño es menor al de dichos parches.

4*.2.1 Prueba estática*

A continuación se muestran los resultados del error entre la posición exacta de la pelota y la recibida por el sistema de Inteligencia Artificial. La posición se da con las coordenadas *(x,y)*.

Cámara 1

Coordenadas reales [cm]	Coordenadas distorsionadas recibidas [cm]	% de error	Coordenadas sin distorsión recibidas [cm]	% de error
(180,130)	(181.02,127.08)	1.26	(179.43,129.42)	0.33
(180,50)	(178.76,43.80)	2.58	(179,50.16)	0.41
(90,115)	(82.59,113.63)	3.08	(88.14,115.30)	0.77
(50,100)	(39.98,95.42)	4.5	(49.21,98.97)	0.53
(50,50)	(43,42.5)	4.19	(49.4,48.3)	0.73
(92,70)	(85.23,62.89)	4.01	(90.5,68.3)	0.92

Cámara 2

Coordenadas reales [cm]	Coordenadas distorsionadas recibidas [cm]	% de error	Coordenadas sin distorsión recibidas [cm]	% de error
(25,108)	(18.6,108.3)	2.57	(22.3,109.6)	1.26
(50,60)	(46.4,57.2)	1.83	(50.26,59.5)	0.23
(99,21)	(97.7,18.25)	1.22	(99.8,20.9)	0.32
(110,70)	(106.9,67.3)	1.65	(109.1,71.82)	0.81
(180,40)	(177.9,35.3)	2.06	(178.1,39.4)	0.79
(160,107)	(159.9,106.3)	0.28	(159.4,106.3)	0.36

4.2.2. Prueba dinámica

A continuación se muestran los resultados del número de paquetes enviados, de paquetes recibidos y de paquetes perdidos.

La tabla siguiente muestra los resultados obtenidos.

Cámara 1

Paquetes enviados	Paquetes recibidos	Paquetes perdidos	% de error
3028	2969	59	1.94
2584	2535	49	1.89
2150	2106	44	2.04
7678	7616	62	0.80

Cámara 2

Paquetes enviados	Paquetes recibidos	Paquetes perdidos	% de error
2840	2840	0	0
4255	4255	0	0
3697	3697	0	0

4.3 Localización de los robots propios

Para la localización de nuestros robots se deben obtener las coordenadas cartesianas de su posición, así como la orientación del robot, para lo cual se utilizan los parches en forma de triángulo [1]. La posición se da de la siguiente forma (x,y,θ).

4.3.1. Prueba estática

A continuación se muestran los resultados del error entre la posición exacta del robot y la recibida por el sistema de Inteligencia Artificial.

Cámara 1

Coordenadas reales [cm]	Coordenadas distorsionadas recibidas [cm]	% de error	Coordenadas sin distorsión recibidas [cm]	% de error
(90,115, 3.1416)	(84,113,3.1416)	2.58	(87.95,115, 3.1416)	0.83
(50,100, 3.1416)	(42.6,93.6, 3.1416)	3.99	(47.3,100, 3.1416)	1.10
(50,50,3.839)	(44.5,42.5,3.894)	3.79	(48.9,48.6,3.85)	0.73
(180,50,3.839)	(178,41.6,3.99)	3.53	(177.1 49.2,3.81)	1.23
(180,130,3.75)	(177.6,127.9,3.8)	1.31	(179.8,129.7,3.71)	0.15
(92,70,4.27)	(87.4, 64.2,4.44)	3.03	(92.3,69,4.18)	0.43

Cámara 2

Coordenadas reales [cm]	Coordenadas distorsionadas recibidas [cm]	% de error	Coordenadas sin distorsión recibidas [cm]	% de error
(50,50,1.57)	(47.5,47.5,1.73)	1.42	(51.4,50.47,1.62)	0.59
(180,50,1.74)	(177.9,45.8,1.57)	1.88	(178.5,49.25,1.81)	0.67
(100,70,4.53)	(97.4,67.34,4.6)	1.49	(100.17,70.6,4.65)	0.25
(25,118,4.53)	(18.6,119.3,4.76)	2.62	(23.4,119,4.44)	0.76
(109,21,3.1416)	(107.2,16.6,3.19)	1.91	(110.2,20.9,3.194)	0.48
(160,117,3.927)	(158.5,116.4,3.88)	0.65	(159.8,115.7,3.88)	0.53

4.3.2. Prueba dinámica

A continuación se muestran los resultados donde se mide el número de veces que el sistema de Inteligencia Artificial deja de detectar a los robots. Al no detectar un robot con el Sistema de Visión y Procesamiento, entonces no se considera válido y por lo tanto no es enviado a la Inteligencia Artificial.

La tabla siguiente muestra los resultados obtenidos.

Cámara 1

Paquetes enviados	Paquetes recibidos	Paquetes perdidos	% de error
7874	5970	1904	24.18
4047	3460	587	14.5
2073	1944	129	6.22
4886	4681	205	4.19

Cámara 2

Paquetes enviados	Paquetes recibidos	Paquetes perdidos	% de error
2134	2133	1	0.04
3506	2825	681	19.42
3010	3005	5	0.16
3693	3620	73	1.97

Los resultados de las pruebas dinámicas para las dos cámaras, donde el porcentaje de error es mayor del 10%, es debido a que para reconocer a nuestros robots se debe de reconocer un parche central que lo distingue del equipo contrario y los parches que forman un triángulo, que son para dar la orientación del robot y el número del robot. A continuación se muestra un robot el cual ha sido identificado por el sistema. La línea que va del centro de la figura hacia la izquierda indica la orientación del robot.

Figura 4.1: Robot detectado por el Sistema de Visión y Procesamiento

Para detectar los parches en el procesamiento, se utiliza la técnica de formación de blobs [1]. En la aplicación de esta técnica, se observa que cuando el robot está en ángulos como 45º, 135º, 225º, 315º, se unen los dos parches de un solo color, por lo que el robot no se detecta. A continuación se muestra un ejemplo de formación de blobs.

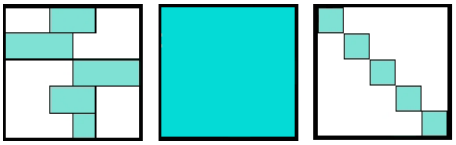


Figura 4.2: Formación de blobs, la imagen de la izquierda y la imagen de la derecha dan como resultado la imagen central

4.4 Localización de los robots contrarios

En esta prueba sólo se detecta la posición de los robots contrarios, a través del parche central, debido a que cada equipo utiliza formas diferentes para obtener la orientación e identificación de sus robots. Por lo tanto la prueba sólo consiste en obtener la posición *(x,y)* del robot en el campo en forma dinámica.

4.4.1 Prueba dinámica

A continuación se muestran los resultados donde se mide el número de veces que el sistema de Inteligencia Artificial deja de detectar a los robots.

La tabla siguiente muestra los resultados obtenidos.

Cámara 1

Paquetes enviados	Paquetes recibidos	Paquetes perdidos	% de error
5531	5531	0	0.0

Cámara 2

Paquetes enviados	Paquetes recibidos	Paquetes perdidos	% de error
1916	1916	0	0.0

Para las dos cámaras se realizaron más pruebas dinámicas de los robots contrarios y el porcentaje de error es igual a los mostrados. Por eso sólo se reporta un resultado por cámara.

4.5 Velocidad de procesamiento

La velocidad máxima de procesamiento que alcanza el sistema es de 30 cps (cuadros por segundo) con una cámara y con las dos cámaras se reparten los 30cps. El software tiene controles que permiten modificar esta velocidad hasta los 50 cps, el rendimiento sólo llega hasta lo mencionado. Y entre más alta sea la resolución de la cámara, mayor es el tiempo de procesamiento. Otros puntos influyentes en la velocidad es el tipo de comunicación entre la cámara y la computadora, y la intensidad de la iluminación en el campo de juego.

CAPÍTULO 5

Conclusiones

El modelo HSI tiene una gran ventaja sobre el modelo RGB. Ésta es que para definir un color en el modelo HSI, se puede hacer sólo con la H, de igual manera que para definir la intensidad, se hace sólo con la I. Mientras que en el modelo RGB el usar sólo una componente no describe nada del color ni de su intensidad.

Para eliminar la distorsión producida por el lente gran angular, existe otro método que es el Método de Tsai. Este método necesita las características extrínsecas e intrínsecas de la cámara dando como resultado una mejor efectividad si es que las características antes mencionadas son exactas. La forma de quitar la distorsión con la ecuación 1.2 es más sencilla en el aspecto de no necesitar características de los dispositivos de captura para eliminar la distorsión, sólo se necesita tener un patrón de rejilla para ir observando el resultado de su aplicación.

Al utilizar las técnicas de reconocimiento de patrones para tomar muestras, se obtuvo la ventaja de poder tener muestras de diferentes partes del campo de juego, esto es importante ya que en los equipos que trabajan con fútbol small size reportan que el cambio de intensidad en el campo de juego es un factor importante para tener un error considerable en la detección de los colores.

Al utilizar dos dispositivos de captura en un solo sistema de visión no se afectó la velocidad de procesamiento del sistema, ya que el procesamiento de las imágenes de estos dispositivos son hechas por 2 instancias que tienen en común la representación y detección de objetos, y sólo varían en las características de distorsión, clasificación de colores y unos parámetros que se usan para escalar las posiciones dadas por el sistema para convertirlas en coordenadas del mundo real.

En el sistema al cual se le incorporaron las modificaciones, mejoró su desempeño. Esto es debido a la utilización del nivel de intensidad, con el cual sólo se transforma de RGB a I el 100% de la imagen y de RGB a HS se transforma aproximadamente un 25% de la imagen, dando como resultado un incremento en la velocidad de procesamiento. También debido a la mejora en el muestreo, se puede, ahora utilizar diversos clasificadores. Al manejar dos dispositivos de captura en un sistema, se mejora la sincronización en el envió de información a la Inteligencia Artificial. Con la corrección de la distorsión se corrige un error en la estimación de la posición, este error está aproximadamente entre un 8% y 10%, además de permitir el uso de las cámaras con lente gran angular, estos lentes ayudan a utilizar menos dispositivos de captura para cubrir el terreno de juego.

La forma de clasificar más útil para el procesamiento que se necesita, es el cuantificador vectorial, debido a que usa la distancia euclidiana para controlar su clasificación (resultados de salida), a esta distancia se le pude variar el rango de pertenencia. La red neuronal sólo cuenta con su salida que va de *[0.0, 1.0]* y esta salida se usa para controlar la clasificación. Si el ambiente cambia demasiado la red neuronal es la que mejor clasifica, pero para detectar los robots y la pelota, es necesario sólo clasificar los datos necesarios y no más, la red neuronal es un buen clasificador cuando se trata de clasificar todo el color que encuentre en una imagen. Además la velocidad al procesar la imagen es más rápida cuando se utiliza el cuantificador vectorial que la red neuronal.

5.1 Trabajo Futuro

Debido a las mejoras antes mencionadas, el desempeño del sistema ha mejorado considerablemente, dando como resultado el poder empezar a desarrollar algún estimador para la posición de objetos dentro del terreno de juego. Uno de las trayectorias más importantes a estimar es la de la pelota.

En este trabajo no se incluyeron mejoras en las etapas finales del procesamiento de la imagen, que son: Reconocimiento de características y reconocimiento de objetos, por lo que habrá que analizar si se puede hacer de un modo más eficiente para acelerar la velocidad de procesamiento y reconocer mejor los objetos para así tener un mejor estimador de posiciones.

También se considera importante investigar la forma de convertir la imagen de RGB a HSI con alguna tarjeta externa (por hardware), ya que con la mejora obtenida en este trabajo, es sólo un respiro mientras no crezcan los elementos a identificar en el terreno de juego, así como el tamaño (resolución) de la imagen.

APÉNDICE A

Descripción de DirectShow

A.1 DirectShow

DirectShow es una arquitectura de Microsoft para audio y video. Provee captura y reproducción de alta calidad para flujos multimedia. Es parte de DirectX9 y está integrado con sus tecnologías, además, detecta y utiliza aceleración de video y audio por hardware cuando se encuentra disponible. Está basado en COM (Component Object Model) que es un modelo de programación orientado a objetos [5].

A.2 Filtros y Pines

DirectShow utiliza una arquitectura modular, donde cada etapa del procesamiento es hecha en un objeto COM llamado filtro. Un filtro es un objeto que realiza alguna operación en el flujo multimedia como las siguientes:

- leer archivos,
- obtener video de un dispositivo de captura,
- decodificar video o pasar datos a la tarjeta gráfica

Los filtros reciben una entrada y generan una salida.

En DirectShow una aplicación realiza una tarea conectando los filtros en cadena, de tal forma que la salida de un filtro sea la entrada de otro. Un conjunto de filtros conectados de esta forma es llamado una gráfica.

Los puntos de conexión entre los filtros son objetos COM y son llamados pines. Los pines se utilizan para mover los datos de un filtro al siguiente.

Los filtros pueden ser divididos en las siguientes categorías [5]:

- Fuente. Un filtro fuente introduce datos en la gráfica. Los datos pueden venir de un archivo, una red, una cámara, etc.
- Transformación. Un filtro de transformación toma un flujo de entrada, procesa los datos y crea un flujo de salida. Codificadores y decodificadores son ejemplos de este tipo de filtros.
- Despliegue. Un filtro de despliegue se encuentra al final de la cadena. Reciben datos y se los presentan al usuario. Por ejemplo, un filtro de despliegue dibuja los cuadros de video en la pantalla.

- Divisor. Un filtro divisor separa el flujo de entrada en dos o más salidas.
- Multiplexor. Toma múltiples entradas y las combina en una sola salida..

Los filtros son controlados por un componente de alto nivel llamado Manejador de la Gráfica de Filtros.

A.2.1 El Manejador de la Gráfica de Filtros.

La aplicación hace llamadas de alto nivel como "Corre" (para empezar a mover los datos a través de la gráfica) o "Detente" (para detener el flujo de datos). Si se requiere más control sobre las operaciones en el flujo, se puede acceder directamente a los filtros utilizando interfaces COM. El Manejador de la Gráfica Filtros manda mensajes de eventos a la aplicación y también proporciona métodos para construir la gráfica, conectando un filtro con otro.

Para escribir una aplicación de DirectShow hay tres pasos que deben ser realizados [5]:

1. La aplicación debe instanciar un Manejador de la Gráfica.
2. La aplicación utiliza el Manejador de la Gráfica para construir una gráfica. El conjunto de filtros en la gráfica depende de la aplicación.
3. La aplicación utiliza el Manejador de la Gráfica para controlar el flujo de datos en los filtros. La aplicación también debe responder a los eventos enviados por el Manejador de la Gráfica.

Cuando el procesamiento concluye, la aplicación debe liberar los recursos del Manejador de la Gráfica y de los demás filtros.

Como se mencionó, DirectShow está basado en COM. El Manejador de la Gráfica y los filtros son objetos COM. La siguiente sección explica su funcionamiento.

A.3 Un objeto COM

Los objetos Component Object Model (COM) son básicamente cajas negras que pueden ser usadas por las aplicaciones para ejecutar una o más tareas. Ellas son más comúnmente implementadas como una librería dinámica (DLL). Como una convencional DLL, los objetos COM exponen métodos que la aplicación puede llamar para ejecutar cualquiera de las tareas soportadas. Las aplicaciones interactúan con los objetos COM.

Los métodos públicos de los objetos COM están agrupados en uno o más interfases. Para usar un método, se debe crear el objeto y obtener la interfase apropiada desde el objeto. Una interfase contiene un conjunto relacionado de métodos que provee acceso a un distintivo particular del objeto.

Se deben usar técnicas específicas del COM para controlar el tiempo de vida del objeto.

Cada objeto COM tiene un único identificador registrado que es usado para crear el objeto. El COM automáticamente carga la DLL correcta.

A.3.1 GUIDs

Los identificadores únicos globales (GUIDs) son una parte clave del modelo de programación del COM. Un GUID es una estructura de 128-bit. Sin embargo, los GUIDs son creados de tal forma que garantizan que dos GUIDs no sean los mismos. COM usa los GUIDs extensivamente para dos propósitos primarios:

- Para identificar únicamente un objeto COM en particular. Un GUID que es asignado a un objeto COM es llamado un identificador de clase (CLSID). Se usa un CLSID cuando se quiere crear una instancia del objeto COM asociado.

- Para identificar únicamente una interfase COM en particular. El GUID que es asociado con una interfase COM en particular es llamado un identificador de interfase (IID). Se usa un IID cuando se requiere de una interfase en particular desde un objeto. Un IID de interfase será el mismo, sin importar cual objeto expone la interfase.

A.3.2 Valores HRESULT

Todos los métodos COM regresan un entero de 32-bit llamado HRESULT. En la mayoría de los métodos, el HRESULT es esencialmente una estructura que contiene 2 piezas primarias de información:

Ya sea que el método tenga éxito o falle.

Algunos métodos regresan valores HRESULT solamente desde el conjunto estándar definido en Winerror.h. Sin embargo, los métodos son libres de regresar los valores acostumbrados con más información especializada.

Mientras los valores de HRESULT son frecuentemente usados para regresar errores de información, no se debe de pensar en ellos como errores de código. El hecho

de que el bit que indica éxito o falla es guardado separadamente desde los bits que contienen la información detallada, permite a los valores de HRESULT tener cualquier número de éxito y fallas de código. Por convención, los códigos de éxito tienen nombres con un prefijo S_, y los códigos de falla con una prefijo E_.

Si se necesita información detallada acerca dela salida de la llamada al método, se necesitará probar cada valor relevante de HRESULT. Sin embargo, se podría estar interesado solamente en si el método tuvo éxito o falló. Una manera robusta de probar si un valor de HRESULT indica éxito o falla es pasar el valor a una de las siguientes macros, definidos en Winerror.h.

La macro SUCCEEDED regresa TRUE por un código de éxito y FALSE por un código de falla. La macro FAILED regresa TRUE por un código de falla y FALSE por un código de éxito .

A.4 Objetos e Interfases

Es importante entender la distinción entre los objetos y las interfases. En un uso casual, un objeto podría algunas veces ser referido por el nombre de su interfase principal. Sin embargo, estrictamente hablando, los 2 términos no son intercambiables
.

- Un objeto podría exponer cualquier número de interfases. Por ejemplo, mientras todos los objetos deben exponer la interfase IUnknown, ellos normalmente exponen al menos una interfase adicional, y ellos podrían exponer muchas más. Para usar un método en particular, no se debe solamente crear el objeto, se debe también obtener la interfase correcta.

- Mas de un objeto podría exponer la misma interfase. Una interfase es un grupo de métodos que ejecutan un conjunto de operaciones especificas. La definición de interfase especifica solamente la sintaxis de los métodos y su funcionalidad general. Cualquier objeto COM que necesita soportar un conjunto particular de operaciones puede hacer eso exponiendo una interfase adecuada. Algunas interfases son altamente especializadas y son expuestas solamente por un solo objeto. Otras son muy usadas en una variedad de circunstancias y son expuestas por muchos objetos. El caso extremo es la interfase IUnknown, la cual debe ser expuesta por todos los objetos COM.

No se puede añadir un nuevo método a una interfase existente. En lugar de esto se debe crear una nueva instancia. Una práctica común es hacer generaciones de nuevas interfases que incluyan todos los métodos de las interfases viejas, mas cualquier nuevo método.

A.5 Creando Objetos COM

Hay muchas maneras para crear objetos COM (Component Object Model). Las 2 más comunes en la programación de Microsoft® DirectX® son:

- Directamente, pasando el identificador de clase del objeto (CLSID) a la función CoCreateInstance. La función creará una instancia del objeto, y ésta regresará un puntero para la interfase que se especificó.
- Indirectamente, llamando al método o función de DirectX que crea el objeto. El método crea el objeto y regresa una interfase sobre el objeto. Cuando se crea un objeto de esta manera, usualmente no se puede especificar a cual interfase debe ser regresado.

Antes de crear cualquier objeto, el COM debe ser inicializado llamando la función CoInitialize. Si se está creando objetos indirectamente, el método para la creación del objeto manejará esta tarea. Si se necesita crear un objeto con CoCreateInstance, se debe llamar explícitamente a CoInitialize. Cuando se está terminando, el COM debe ser desinicializado llamando CoUninitialize**.** Por lo que **s**i se hace una llamada a CoInitialize debe ir acompañada con una llamada a CoUninitialize.

Para crear una nueva instancia de un objeto COM con CoCreateInstance, se debe tener los CLSID de los objetos.

La función CoCreateInstance tiene 5 parámetros. Para los objetos COM que se usan con DirectX, se pueden poner los parámetros como sigue:

- *rclsid*. Se coloca este parámetro para el CLSID del objeto que se quiere crear.
- *pUnkOuter*. Se coloca este parámetro en NULL. Este es usado solamente si se están agregando objetos.
- *dwClsContext*. Se coloca este parámetro para CLSCTX_INPROC_SERVER. Este indica que el objeto es implementado como una librería dinámica (DLL) y correrá como parte del proceso de la aplicación.
- *riid*. Se coloca este parámetro para el IID de la interfase que se quiera tener de regreso. La función creará el objeto, y regresará el puntero de la interfase requerida en el parámetro *ppv*.
- *ppv*. Se coloca este parámetro para la dirección de un puntero que será puesto para la interfase especificada por *riid* cuando la función regrese. Esta variable deberá ser declarada como un puntero para la interfase requerida.

A.5.1 Usando las interfaces COM

Cuando un objeto COM es creado, la creación del método regresa un puntero de interfase. Se puede entonces usar ese puntero para acceder cualquier interfase de los métodos.

```
CoInitialize( NULL );
...
hr = CoCreateInstance( CLSID_DirectPlay8Peer, NULL,CLSCTX_INPROC_SERVER,
                    IID_IDirectPlay8Peer, (LPVOID*) &g_pDP );
hr = g_pDP->Initialize( NULL, DirectPlayMessageHandler, 0 );
```

A.6 Cargando una Gráfica desde un proceso externo

GraphEdit puede cargar una gráfica de filtros creada por un proceso externo. Con este distintivo, se puede ver exactamente que gráfica de filtros construye la aplicación, con sólo una cantidad mínima de código en la aplicación.

La aplicación debe registrar la instancia de la gráfica de filtros en la Running Object Table (ROT). La ROT es una tabla global de búsqueda accesible que sigue la pista de los objetos que están corriendo. Los objetos están registrados en el ROT por el moniker. Para conectarte a la gráfica, GraphEdit busca la ROT por monikers los cuales despliegan el nombre junto con un formato en particular:

!FilterGraph *X* pid *Y*

donde *X* es la dirección hexadecimal del Manejador de la Gráfica de Filtros, y *Y* es el identificador de proceso en hexadecimal.

Cuando la primer aplicación crea la gráfica de filtro, llama la siguiente función:

```
HRESULT AddToRot(IUnknown *pUnkGraph, DWORD *pdwRegister)
{
   IMoniker * pMoniker;
   IRunningObjectTable *pROT;
   if (FAILED(GetRunningObjectTable(0, &pROT))) {       return E_FAIL;    }
   WCHAR wsz[256];
   wsprintfW(wsz, L"FilterGraph %08x pid %08x", (DWORD_PTR)pUnkGraph, GetCurrentProcessId());
   HRESULT hr = CreateItemMoniker(L"!", wsz, &pMoniker);
   if (SUCCEEDED(hr)) {
      hr = pROT->Register(ROTFLAGS_REGISTRATIONKEEPSALIVE, pUnkGraph,
         pMoniker, pdwRegister);
      pMoniker->Release();
   }
   pROT->Release();
   return hr;
```

```
}
```

Esta función crea un moniker y una nueva entrada en la ROT para la gráfica de filtros. El primer parámetro es un puntero a la gráfica de filtros. El segundo parámetro recibe un valor que identifica la nueva entrada en la ROT. Antes de que la aplicación suelte la gráfica de filtros, se llama a la siguiente función para remover la entrada de la ROT. El parámetro *pdwRegister* es el identificador regresado por la función AddToRot.

```
void RemoveFromRot(DWORD pdwRegister)
{
   IRunningObjectTable *pROT;
   if (SUCCEEDED(GetRunningObjectTable(0, &pROT))) {
      pROT->Revoke(pdwRegister);
      pROT->Release();
   }
}
```

A.7 Usando el Sistema Enumerador de Dispositivos

El sistema enumerador de dispositivos provee una manera uniforme de enumerar, por categoría, los filtros registrados en el sistema del usuario. Además, éste diferencia entre los dispositivos de hardware individuales, aún cuando el mismo filtro los soporte. Este es particularmente muy usado para dispositivos que usan el Windows Driver Model (WDM) y el filtro KSProxy. Por ejemplo, el usuario podría tener muchos dispositivos de captura de video WDM, todos soportados por el mismo filtro. El Sistema enumerador de dispositivos los trata como instancias de dispositivos separados.

El Sistema enumerador de dispositivos trabaja creando un enumerador para una categoría específica, tal como captura de audio o compresión de video. El enumerador de categoría regresa un moniker único para cada dispositivo en la categoría. El enumerador de categoría automáticamente incluye cualquier dispositivo relevante Plug and Play en la categoría.

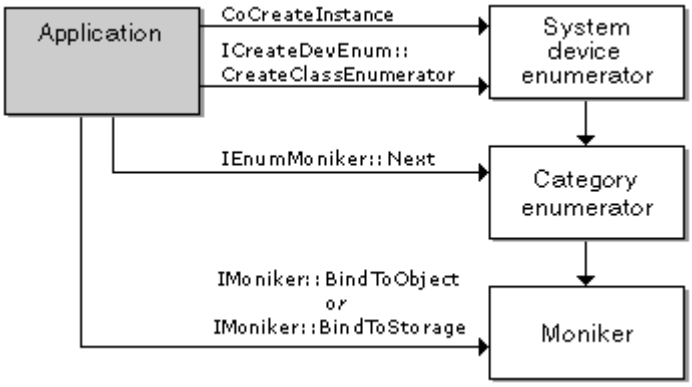
Para usar el Sistema enumerador de dispositivos, se hace lo siguiente:

1. Se crea el sistema enumerador de dispositivos llamando CoCreateInstance. El identificador de clase es CLSID_SystemDeviceEnum.
2. Se obtiene un enumerador de categoría llamando IcreateDevEnum::CreateClassEnumerator con el CLSID de la categoría deseada. Este método regresa un puntero de interfase IEnumMoniker. Si la categoría está vacía (o no existe), el método regresa S_FALSE así como un código de error. Por eso, explícitamente se prueba por S_OK cuando se llama a CreateClassEnumerator, en lugar de llamar a la macro SUCCEEDED.
3. Usa el método IEnumMoniker::Next para enumerar cada moniker. Este método regresa un puntero de interfase IMoniker. Cuando el método Next alcanza el fin

de la enumeración, éste también regresa S_FALSE, por lo que checa de nuevo por un S_OK.

4. Para recuperar el nombre amigable del dispositivo, llamar al método IMoniker::BindToStorage.
5. Para crear e inicializar el filtro de DirectShow que maneja los dispositivos, llamar IMoniker::BindToObject en el moniker. Llamar IFilterGraph::AddFilter para añadir el filtro a la gráfica.

El siguiente diagrama ilustra este proceso.



A.8 Seleccionando el Dispositivo de Captura

Para seleccionar un dispositivo de captura, se usa el Sistema Enumerador de Dispositivos. Este objeto de ayuda regresa una colección de monikers de los dispositivos, seleccionados por categoría de filtros. Un moniker es un objeto COM que contiene información acerca de otro objeto, el cual habilita la aplicación para conseguir información acerca del objeto sin crear el objeto él mismo. Después, la aplicación puede usar el moniker para crear el objeto.

Cuando se selecciona un dispositivo, se debe crear el filtro de captura para el dispositivo llamando a IMoniker::BindToObject en el moniker. Entonces llama AddFilter para añadir el filtro a la gráfica de filtros:

A.9 El Método AddFilter

El método AddFilter añade un filtro a la gráfica. Su sintaxis es la siguiente:

```
HRESULT AddFilter( IBaseFilter *pFilter, LPCWSTR pName );
```

donde

pFilter es un puntero para la interfase IBaseFilter del filtro añadido.
pName es un puntero para una cadena de caracteres que contiene un nombre para el filtro.

A.10 Construyendo Gráficas con el Constructor de la Gráfica de Captura

A pesar de su nombre, el Constructor de la Gráfica de Captura es muy usado para construir muchos tipos de graficas de filtros comunes, no solamente gráficas de captura.

El Constructor de la Gráfica de Captura expone la interfase ICaptureGraphBuilder2. Empieza llamando a CoCreateInstance para crear el Constructor de la Gráfica de Captura y el Manejador de la Gráfica de Filtros. Entonces inicializa el Constructor de la Gráfica de Captura llamando a ICaptureGraphBuilder2::SetFiltergraph con un puntero a el Manejador de la Gráfica de Filtros, como sigue:

```
IGraphBuilder *pGraph = NULL;
ICaptureGraphBuilder2 *pBuilder = NULL;
// Create the Filter Graph Manager.
HRESULT hr =  CoCreateInstance(CLSID_FilterGraph, NULL,
   CLSCTX_INPROC_SERVER, IID_IGraphBuilder, (void **)&pGraph);

if (SUCCEEDED(hr))
{
   // Create the Capture Graph Builder.
   hr = CoCreateInstance(CLSID_CaptureGraphBuilder2, NULL,
      CLSCTX_INPROC_SERVER, IID_ ICaptureGraphBuilder2, (void **)&pBuilder);
   if (SUCCEEDED(hr))    pBuilder->SetFiltergraph(pGraph);
};
```

A.10.1 Conectando Filtros

El método ICaptureGraphBuilder2::RenderStream conecta 2 o 3 filtros juntos en una cadena. Generalmente, el método trabaja mejor cuando cada filtro no tiene más que un pin de entrada o de salida del mismo tipo. Esta explicación inicia ignorando los primeros 2 parámetros de RenderStream y se enfoca en los últimos 3 parámetros. El tercer parámetros es un puntero IUnknown, el cual puede especificar un filtro (como un puntero interfase IBaseFilter) o un pin de salida (como un puntero interfase IPin). El cuarto y quinto parámetro especifican punteros IBaseFilter. El método RenderStream conecta los tres filtros en una cadena. Por ejemplo, suponga que A, B y C son filtros y se asume que cada filtro tiene exactamente un pin de entrada y un pin de salida. La siguiente llamada conecta A a B, y entonces B a C:

```
RenderStream(NULL, NULL, A, B, C)
```

Todas las conexiones son “inteligentes”, esto significa que los filtros adicionales son añadidos a la gráfica como se necesite. Para conectar solo 2 filtros, se coloca en medio el valor de NULL. Por ejemplo, esta llamada conecta A a C:

```
RenderStream(NULL, NULL, A, NULL, C)
```

Se pueden crear largas cadenas llamando al método 2 veces:

```
RenderStream(NULL, NULL, A, B, C)
RenderStream(NULL, NULL, C, D, E)
```

Si el último parámetro es NULL, el método automáticamente localiza a un intérprete. Este usa el Video Renderer para video y el DirectSound Renderer para audio. De este modo:

RenderStream(NULL, NULL, A, NULL, NULL) es equivalente a
RenderStream(NULL, NULL, A, NULL, R)

donde *R* es el apropiado intérprete.

Si se especifica un filtro en el tercer parámetro, mas bien un pin, se puede necesitar indicar cual pin de salida debe ser usado para la conexión. Este es el propósito de los 2 primeros parámetros del método. El primer parámetro aplica solamente para el filtro de captura. Esto lo especifica una GUID que indica una categoría de pin. Dos de estas categorías son válidas para todos los filtros de captura:

```
PIN_CATEGORY_CAPTURE
PIN_CATEGORY_PREVIEW
```

Para un archivo de captura, se conecta el pin de captura a un filtro multiplexor. Para mantener el preview, conecta el pin preview a un interprete. Si se escogen las dos categorías, la gráfica podría tirar un número excesivo de frames durante la captura del archivo; pero si la gráfica está conectada apropiadamente, ésta tira tantos frames del preview como se necesiten para mantener continuidad en el flujo de captura.

El siguiente ejemplo muestra como conectar ambos flujos:

```
// Capture to file:
pBuilder->RenderStream(&PIN_CATEGORY_CAPTURE, NULL, pCapFilter, NULL, pMux);
// Preview:
pBuilder->RenderStream(&PIN_CATEGORY_PREVIEW, NULL, pCapFilter, NULL, NULL);
```

El segundo parámetro para identificar a RenderFile el tipo de media, es uno de los siguientes:

MEDIATYPE_Audio
MEDIATYPE_Video
MEDIATYPE_Interleaved (DV)

A.10.2 Encontrando Interfases en Filtros y Pines

Después de construir una gráfica, se necesita localizar varias interfases expuestas por los filtros y los pins en la gráfica.

La manera más simple para encontrar una interfase es usar el método ICaptureGraphBuilder2::FindInterface. Este método pasea en la gráfica (filtros y pines) hasta localizar la interfase deseada. Se puede especificar el punto inicial para buscar, y se puede limitar la búsqueda a los filtros con el flujo hacia arriba o el flujo hacia abajo desde el punto inicial.

A.10.3 Encontrando Pines

Se puede necesitar localizar un pin individual en un filtro, aunque en la mayoría de los casos los métodos RenderStream y FindInterface resolverán el problema. Si se necesita encontrar un pin en particular en un filtro, el método de ayuda ICaptureGraphBuilder2::FindPin es muy usado. Se especifica la categoría, el tipo de media (video o audio), la dirección, y si el pin debe estar desconectado.

A.11 Cuando un evento ocurre

Para procesar los eventos de DirectShow, una aplicación necesita una manera para encontrar una salida cuando los eventos están esperando en la cola. El Manejador de la Gráfica de Filtros provee 2 maneras para hacer esto:

- Window notification: El Manejador de la Gráfica de Filtros envía un mensaje a un usuario definido de Windows para una aplicación de ventana si hay un nuevo evento.
- Event signaling: El Manejador de la Gráfica de Filtros señala un evento de Windows si hay eventos de DirectShow en la cola, y vuelve a iniciar los eventos si la cola esta vacía.

Una aplicación puede usar cualquier técnica. La más simple es usualmente Window notification.

A.11.1 Window Notification

Para iniciar window notification, se llama al método IMediaEventEx::SetNotifyWindow y se especifica una mensaje privado. Las aplicaciones pueden usar mensajes numéricos en el rango de WM_APP a 0xBFFF como mensajes privados. Siempre que el Manejador de la Gráfica de Filtros pone una nueva notificación de evento en la cola, éste envía el mensaje a la ventana designada. La aplicación responde al mensaje desde adentro del ciclo de mensaje de la ventana.

El siguiente código de ejemplo muestra cómo colocar la notificación de ventana.

```
#define WM_GRAPHNOTIFY WM_APP + 1   // Private message.
pEvent->SetNotifyWindow((OAHWND)g_hwnd, WM_GRAPHNOTIFY, 0);
```

El mensaje es un ordinario mensaje de Windows, y es enviado separadamente desde la cola de notificación de evento de DirectShow. La ventaja de esta aproximación es que la mayoría de las aplicaciones ya implementan un ciclo de mensajes. Por eso, se puede incorporar el manejo de evento de DirectShow sin mucho trabajo adicional.

El siguiente código de ejemplo muestra un bosquejo de como responder a la notificación de mensaje.

```
LRESULT CALLBACK WindowProc( HWND hwnd, UINT msg, UINT wParam, LONG lParam)
{
   switch (msg)
   {
      case WM_GRAPHNOTIFY:
         HandleEvent();  // Application-defined function.
         break;
      // Handle other Windows messages here too.
   }
   return (DefWindowProc(hwnd, msg, wParam, lParam));
}
```

Como la notificación del evento y el ciclo del mensaje son ambos asíncronos, la cola podría contener más de un evento en el tiempo en el que la aplicación responde al mensaje. También, los eventos pueden ser limpiados algunas veces de la cola si ellos llegan a ser inválidos. Por tanto, en el código de manejo de evento, llama a GetEvent hasta que éste regrese un código de falla, indicando que la cola esta vacía.

Antes de liberar el puntero IMediaEventEx, se cancela la notificación del evento llamando a SetNotifyWindow con un puntero en NULL. En el código para procesar el evento, checa si el puntero IMediaEventEx es valido antes de llamar a GetEvent. Estos pasos previenen un posible error, en el cual la aplicación recibe la notificación del evento después de que se ha soltado el puntero IMediaEventEx.

A.12 Algoritmo para usar 1 o más dispositivos de video en un sistema

El siguiente algoritmo es para desplegar ventanas que muestran los diferentes dispositivos de captura de video.

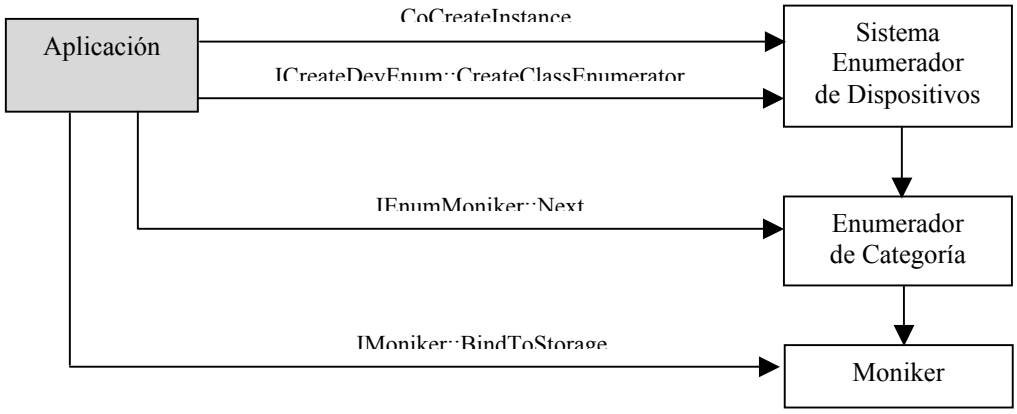
-Se inicializa el COM con CoInitialize

-Se crea el Manejador de la Gráfica de Filtros con CoCreateInstance para tener el objeto de gráfica de filtros (CLSID_FilterGraph) y la interfase constructora de gráficas (IID_IGraphBuilder)

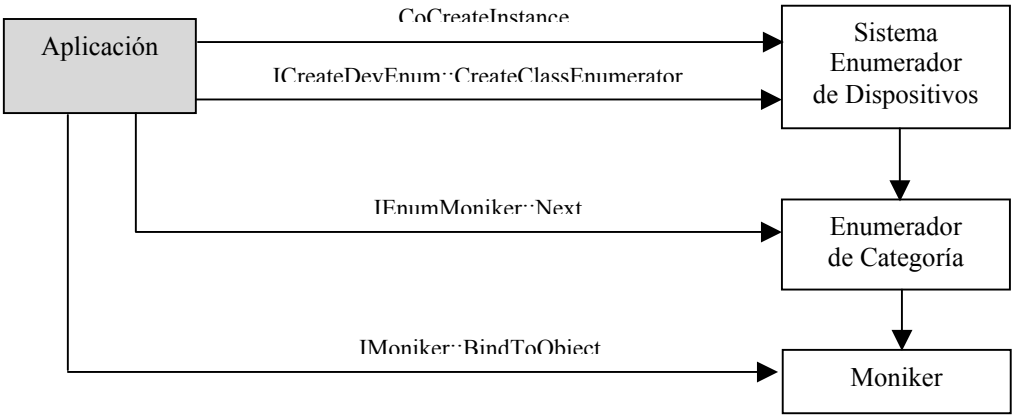
-Se crea un moniker y una nueva entrada en la ROT para la gráfica de filtros. Esto se realiza llamando a AddToRot.

-Se crea el Sistema Enumerador de Dispositivos

Para tomar la descripción del dispositivo, el nombre amigable del dispositivo y el número de dispositivos. Enumerando la categoría de dispositivos video con CLSID_VideoInputDeviceCategory. Esto se realiza como lo muestra la figura.



-Se crea e inicializa el filtro de DirectShow que maneja los dispositivos. Esto se realiza como lo muestra la figura.



-Se añade el filtro a la gráfica a través de AddFilter.

-Soltamos el Sistema Enumerador de Dispositivos

-Se crea el Constructor de la Gráfica de Captura con CoCreateInstance y se inicializa el Constructor de la Gráfica de Captura llamando a SetFiltergraph con un puntero a el Manejador de la Gráfica de Filtros

-Se conecta un pin de salida en un filtro fuente, se indica la categoría del pin con la GUID PIN_CATEGORY_PREVIEW y el método localiza a un interprete.

-Se busca la interfase que implementa el Manejador de la Gráfica de Filtros para controlar el flujo de datos. Se corren todos los filtros en la gráfica de filtros.

-Se sueltan todas las interfases

-Se remueve la entrada de la ROT.

-Se suelta la gráfica de filtros

-Se desinicializa el COM llamando a CoUninitialize

Para usar este algoritmo en aplicaciones de ventana, hay que agregar código para el manejo de despliegue de la imagen en una ventana, especificar bien el tipo de media que se requiere, las conexiones y búsquedas de los pines, y el manejo de eventos en DirectShow.

El manejo de evento que se ocupa es Window Notification y lo que hace es que recibe el mensaje (WM_CAPTURE_BITMAP). Cuando esto ocurre la notificación recibe todos los eventos de este tipo que están en la cola y la aplicación responde al mensaje desde adentro del ciclo de mensaje de la ventana.

A.13 Interfaces soportadas por DisrectShow

A.13.1 La interfase IUnknown

Todos los objetos COM soportan una interfase llamada IUnknown. Esta interfase provee control del tiempo de vida de los objetos y la habilidad para recuperar otras interfases implementadas por el objeto. IUnknown tiene 3 métodos.

AddRef	Incrementa el contador de referencia a interfases en 1.
QueryInterface	Determina si el objeto soporta una interfase COM en particular. Si lo hace, el sistema incrementa el contador de referencia a objetos, y la aplicación puede usar esta interfase inmediatamente.
Release	Decrementa el contador de referencia de la interfase en 1.

Después de que el contador de referencia a objetos llega a 0, el objeto es destruido, y todas las interfases a este objeto llegan a ser inválidas.

El método IUnknown::QueryInterface determina si un objeto soporta una interfase específica. Si un objeto soporta una interfase, IUnknown::QueryInterface regresa un puntero a esa interfase. Entonces puedes usar los métodos de esa interfase para comunicarse con el objeto. Si es exitoso IUnknown::QueryInterface éste llama implícitamente a IUnknown::AddRef para incrementar el contador de referencia, por lo que la aplicación debe llamar a IUnknown::Release para decrementar el contador de referencia antes de destruir el puntero a la interfase.

A.13.2 La interfase IGraphBuilder

Esta interfase provee métodos que habilitan a una aplicación para construir una gráfica de filtro. El Manejador de la Gráfica de Filtros (Filter Graph Manager) implementa esta interfase.

IGraphBuilder hereda desde la interfase IFilterGraph. IFilterGraph provee operaciones básicas, tales como añadir un filtro a la gráfica o conector 2 pines. IGraphBuilder añade métodos adicionales que construyen graficas desde información parcial. Por ejemplo, el método IGraphBuilder::RenderFile construye una gráfica para

reproducir un archivo, dando el nombre del archivo. El método IGraphBuilder::Render interpreta datos desde un pin de salida conectando nuevos filtros a el pin.

Usando estos métodos, una aplicación no necesita especificar cada conexión de filtro y de pin en la gráfica. En lugar de ello, el Manejador de la Gráfica de Filtros selecciona los filtros que están registrados en el sistema del usuario, los añade a la gráfica, y los conecta.

Esto son algunos métodos expuestos por la interfase IGraphBuilder.

Connect	Conecta 2 pines. Si no están conectados directamente, los conecta.
Render	Añade una cadena de filtros a un pin de salida específico para ejecutarlo.
RenderFile	Construye una gráfica de filtros que ejecuta el archivo especificado.
AddSourceFilter	Añade un filtro fuente para la gráfica de filtros para un archivo específico.
Abort	Pide que el constructor de la gráfica regrese lo más rápido posible de su tarea actual.

A.13.3 La interfase IBaseFilter

La interfase IBaseFilter provee métodos para controlar un filtro. Todos los filtros de DirectShow exponen esta interfase. El Manejador de la Gráfica de Filtros usa esta interfase para controlar los filtros. Las aplicaciones pueden usar esta interfase para enumerar los pines y buscar por información del filtro, pero no se debe usar para cambiar el estado de un filtro. En lugar de ello, hay que usar la interfase IMediaControl en el Manejador de la Gráfica de Filtros.

En adición a los métodos heredados desde IMediaFilter, la interfase IBaseFilter expone los siguientes métodos.

EnumPins	Enumera los pines sobre este filtro.
FindPin	Recupera el pin con el identificador especificado.
QueryFilterInfo	Recupera información acerca del filtro.

A.13.4 La interfase IMediaControl

La interfase IMediaControl provee métodos para controlar el flujo de datos a través de la gráfica de filtros. Esto incluye métodos para correr, pausar, y parar la gráfica. El Manejador de la Gráfica de Filtros implementa esta interfase.

En adición a los métodos heredados desde IDispatch, la interfase IMediaControl expone los siguientes métodos.

Run	Corre todos los filtros en la gráfica de filtros.
Pause	Pausa todos los filtros en la gráfica de filtros.
Stop	Detiene todos los filtros en la gráfica de filtros.
StopWhenReady	Pausa la gráfica de filtros, permitiendo a los filtros encolar datos, y entonces detiene la gráfica de filtros.
GetState	Recupera el estado de la gráfica de filtros.

A.13.5 La interfase ICaptureGraphBuilder2

La interfase ICaptureGraphBuilder2 provee métodos para construir gráficas de captura. El objeto Capture Graph Builder implementa esta interfase.

La interfase ICaptureGraphBuilder2 expone los siguientes métodos.

AllocCapFile	Preasigna un archivo de captura a un tamaño especificado.
ControlStream	Coloca el tiempo inicial y final para uno o más flujos de datos capturados.
CopyCaptureFile	Copia los datos de media válidos desde un archivo de captura.
FindInterface	Busca la gráfica para una interfase específica, empezando desde un filtro especificado.
FindPin	Recupera un pin particular en un filtro, o determina si un pin dado pertenece al criterio especificado.
GetFiltergraph	Recupera la gráfica de filtros que el constructor está usando.

RenderStream	Conecta un pin de salida en un filtro fuente a un filtro recuperado, opcionalmente a través de un filtro de compresión.
SetFiltergraph	Especifica una gráfica de filtros para que la use el constructor de gráficas de captura.
SetOutputFileName	Crea la sección escrita del archivo de la gráfica de filtros.

A.13.6 La Interfase ISampleGrabber

La interfase ISampleGrabber es expuesta por Sample Grabber Filter. Ésta habilita una aplicación a recuperar unas muestras de media individuales a medida que se mueven a través de la gráfica de filtros.

La interfase ISampleGrabber expone los siguientes métodos.

SetOneShot	Especifica si el filtro debe detener la gráfica después de recibir una muestra.
SetMediaType	Especifica el tipo de media para la conexión en el pin de entrada del Sample Grabber.
GetConnectedMediaType	Recupera el tipo de media para la conexión en el pin de entrada del Sample Grabber.
SetBufferSamples	Especifica que para copiar la muestra de datos en un bufer a medida que éste va a través del filtro.
GetCurrentBuffer	Recupera una copia de la muestra que el filtro recibió más recientemente.
SetCallback	Especifica un método para retirar para llamar unas muestras entrantes.

A.13.7 La Interfase ICreateDevEnum

La interfase ICreateDevEnum crea un enumerador para dispositivos dentro de una categoría en particular, tales como dispositivos de captura de video, dispositivos de captura de audio, compresores de video, etc. El System Device Enumerator expone esta interfase.

Las aplicaciones pueden usar esta interfase para enumerar dispositivos y crear filtros en DirectShow que manejan cada dispositivo. El método CreateClassEnumerator regresa un enumerador de objetos para una categoría específica del dispositivo. El

enumerador de objetos soporta la interfase IEnumMoniker y regresa una lista de monikers, donde cada moniker representa un dispositivo diferente. En algunos casos, el mismo filtro de DirectShow maneja un categoría entera de dispositivos.

La interfase ICreateDevEnum expone el siguiente método.

CreateClassEnumerator	Crea un enumerador de clase para una categoría específica del dispositivo.

A.13.8 La Interfase IVideoWindow

La interfase IVideoWindow coloca propiedades en la ventana de video. Las aplicaciones pueden usar esto para colocar el origen de la ventana, la posición y dimensiones de la ventana, y otras propiedades.

Las aplicaciones deben usar la versión del Manejador de la Gráfica de Filtros de esta interfase. El Manejador de la Gráfica de Filtros envía a todos los métodos a llamar a Video Renderer. También envía un cierto mensaje a la ventana, como WM_DISPLAYCHANGE, que requiere al Video Renderer para actualizarse. Sin embargo, si la gráfica de filtros contiene más de un Video Renderer, el Manejador de la Gráfica de Filtros solamente comunica con uno de ellos (seleccionado arbitrariamente). Por eso, para trabajar con múltiples ventanas de video, una aplicación debe usar la interfase IVideoWindow directamente en los filtros. En este caso, se debe estar seguro de enviar un mensaje a la ventana para cada Video Renderer , usando el método IVideoWindow::NotifyOwnerMessage.

Las propiedades puestas en un intérprete de video persisten entre sucesivas conexiones y desconexiones.

En adición a los métodos heredados desde IDispatch, la interfase IVideoWindow expone los siguientes métodos.

GetMaxIdealImageSize	Recupera el tamaño máximo ideal para la imagen de video.
GetMinIdealImageSize	Recupera el tamaño mínimo ideal para la imagen de video.
GetRestorePosition	Recupera la posición restaurada de la ventana.
GetWindowPosition	Recupera la posición de la ventana de video.
NotifyOwnerMessage	Envía un mensaje a la ventana de video.

put_AutoShow	Especifica si el intérprete de video automáticamente muestra la ventana de video cuando recibe los datos de video.
put_FullScreenMode	Habilita o deshabilita el modo full-screen.
put_Height	Coloca la altura de la ventana de video.
put_Left	Coloca la coordenada-x de la ventana de video.
put_MessageDrain	Especifica una ventana para recibir mensajes del ratón y el teclado desde la ventana de video.
put_Owner	Especifica una ventana origen para la ventana de video.
put_Top	Coloca la coordenada-y de la ventana de video.
put_Visible	Muestra u oculta la ventana de video.
put_Width	Coloca el ancho de la ventana de video.
put_WindowState	Muestra, oculta, minimiza, o maximiza la ventana de video.
put_WindowStyle	Coloca el estilo de ventana sobre la ventana de video.
SetWindowPosition	Coloca la posición de la ventana de video.

A.13.9 Preguntando por Interfases Adicionales

En muchos casos, el puntero de interfase que se recibe desde la creación del método podría ser lo único que se necesite. En realidad, esto es relativamente común para un objeto el exportar solamente una interfase más que IUnknown. Sin embargo, muchos objetos exportan múltiples interfases, y se podrían necesitar punteros para varios de ellos. Si se necesitan mas interfases que la única regresada por la creación del método, no hay necesidad para crear un nuevo objeto. Simplemente se pide otro puntero de interfase usando el método IUnknown del objeto.

Si se crea el objeto con CoCreateInstance, puede requerir un puntero de interfase IUnknown, y entonces llamar IUnknowm::QueryInterface para requerir cada interfase que se necesite. Sin embargo, esta aproximación es inconveniente si se necesita solamente una sola interfase, y esta no trabaja del todo si se usa un método de creación de objeto que no permite que se especifique cual puntero de interfase debe de ser regresado. En la práctica, usualmente no se necesita obtener un puntero IUnknown explícito porque todas las interfaces COM extienden de la interfase IUnknown.

Extender una interfase es similar a heredar desde una clase en C++. La interfase hija expone todas las interfases de los métodos de los padres, más una o más de su propiedad. Se necesita recordar que la herencia es interna a el objeto. La aplicación no

puede heredar desde o extender la interfase del objeto. Sin embargo, se puede usar la interfase hija para llamar a cualquiera de los métodos de los hijos o de los padres.

Porque todas las interfases son hijas de IUnknown se puede usar cualquiera de los punteros de interfase que ya se tiene para el objeto para llamar QueryInterface. Cuando se hace eso, se debe proveer el identificador de interfase (IID) de la interfase que se está requiriendo y la dirección del puntero que contendrá el puntero de interfase cuando el método regresa.

A.13.10 Categorías para los dispositivos de captura

Para los dispositivos de captura, las siguientes categorías son relevantes.

Categoría GUID	**Descripción**
CLSID_AudioInputDeviceCategory	Dispositivos de captura de Audio
CLSID_VideoInputDeviceCategory	Dispositivos de captura de Video

Un dispositivo puede aparecer en ambas categorías. El siguiente código crea un enumerador para dispositivos de captura de video:

```
ICreateDevEnum *pDevEnum = NULL;
IEnumMoniker *pEnum = NULL;

// Create the System Device Enumerator.
HRESULT hr = CoCreateInstance(CLSID_SystemDeviceEnum, NULL,
    CLSCTX_INPROC_SERVER, IID_ICreateDevEnum,
    reinterpret_cast<void**>(&pDevEnum));
if (SUCCEEDED(hr))
{
    // Create an enumerator for the video capture category.
    hr = pDevEnum->CreateClassEnumerator(
        CLSID_VideoInputDeviceCategory,
        &pEnum, 0);
}
```

La interfase IEnumMoniker regresa una lista de interfases IMoniker, cada una de las cuales representa un moniker de dispositivo. Llama al método IMoniker::BindToStorage, el cual regresa un puntero de interfase IPropertyBag, y entonces llama a IPropertyBag::Read para leer las propiedades desde el moniker. Las propiedades incluidas son las siguientes:

Propiedad	Descripción
FriendlyName	El nombre del dispositivo.
Description	Una descripción del dispositivo.
DevicePath	Una cadena única.

La propiedad del FriendlyName está disponible para cada dispositivo. Este contiene un nombre del dispositivo que sea entendible al humano.

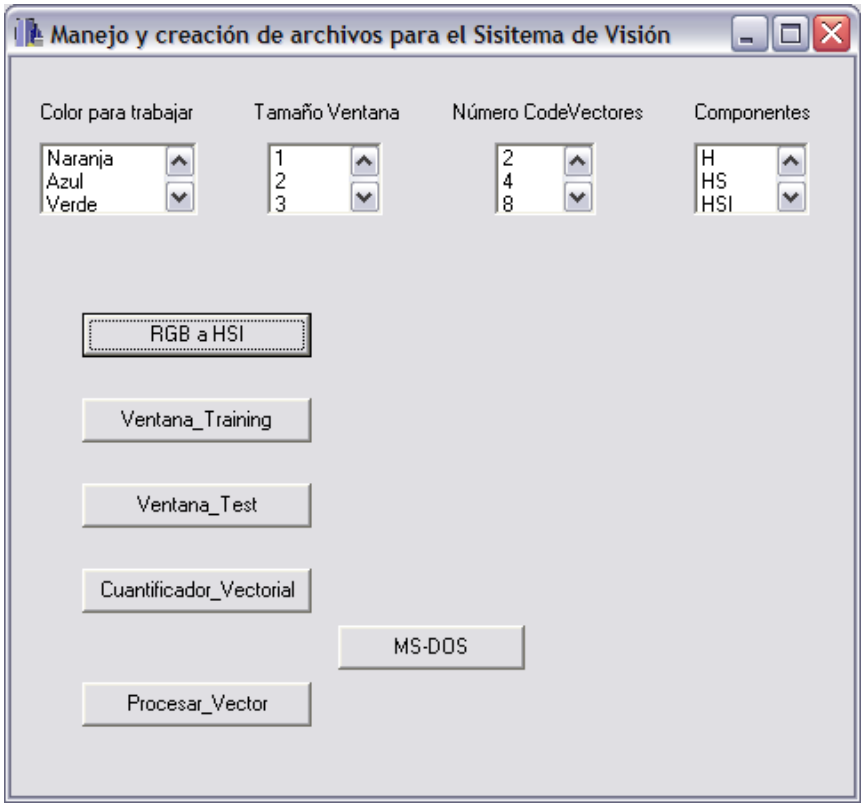
La propiedad de la Description contiene una descripción del dispositivo el cual es más específico que la propiedad del nombre FriendlyName. Generalmente incluye el nombre del vendedor.

La propiedad DevicePath no es una cadena legible para el humano, pero garantiza ser única por dispositivo. Se puede usar esta propiedad para distinguir entre 2 o más instancias del mismo modelo del dispositivo.

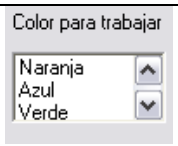
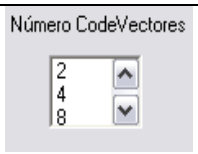
APÉNDICE B

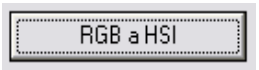
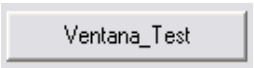
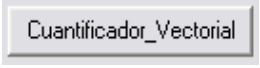
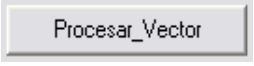
Manejo y creación de archivos para el Sistema de Visión.

El siguiente programa se usa para manejar las imágenes capturadas por el Sistema de Calibración. Este manejo de archivos es muy práctico, ya que no es necesario tener conectada la cámara para poder usar la información recopilada para ésta y hacer pruebas de procesamiento de la imagen.



Estas imágenes se guardan en un formato de matriz, en el encabezado del archivo se escriben las dimensiones (x,y) de la imagen y después los datos de cada píxel en el formato B,G,R.

Control de Listas	*Descripción*
	La lista "Color para trabajar" se usa para escoger el archivo que se va a abrir, se usan 6 tipos de archivos, uno por cada color, que son Naranja, Azul, Verde, Amarillo, Rosa y Cyan.
	La lista de "Tamaño Ventana" es para elegir como usar los píxeles para entrenar la red neuronal. El tamaño de ventana de 1, es para un sólo píxel; el tamaño de ventana de 2, es para 2*2=4 pixeles y así para cada elemento de la lista.
	La lista "Número CodeVectores" es para escoger cuántos codevectores tendrá el codebook de cada color. El tamaño disponible de codevectores está en potencia de 2 ya que se usa el Método de Splitting para la cuantificación vectorial.
	La lista de "Componentes" es para elegir las componentes a usar para entrenar y probar tanto la red neuronal como el cuantificador vectorial.

Control de Botones	*Descripción*
	El botón "RGB a HSI" crea un archivo el cual guarda en un formato de matriz, en el encabezado del archivo se escriben las dimensiones (x,y) y después los datos de cada píxel en el formato H,S,I.
	El botón de "Ventana_Training" crea archivos para utilizar en el entrenamiento de la red neuronal y el cuantificador vectorial, estos archivos contienen los valores de los píxeles de acuerdo con lo elegido en las listas anteriormente mencionadas.
	El botón de "Ventana_Test" crea archivos para probar la red neuronal y el cuantificador vectorial, estos archivos contienen los valores de los píxeles en el orden elegido de las listas antes mencionadas.
	El botón "Cuantificador_Vectorial" abre los archivos creados para el entrenamiento y realiza la cuantificación vectorial. El resultado lo guarda en un archivo. Se guarda un archivo por cada color y contiene el número de codevectores indicados. Por lo que el archivo es el codebook del color correspondiente.
	El botón "Procesar_Vector" utiliza los archivos creados en el entrenamiento del cuantificador vectorial y revisa a qué color pertenece el archivo de prueba indicado.
	El botón "MS-DOS también puede mandar llamar lo que realiza el "Cuantificador_Vectorial" y el "Procesar_Vector". La ventaja es que va desplegando los procesos que realiza.

BIBLIOGRAFÍA

[1] Francisco Javier Rodríguez García
"Sistema de Reconocimiento de Patrones usando Procesamiento de Imágenes para Localización de Robots Móviles"
Tesis UNAM, México, 2007

[2] Luis Alfredo Martínez Gómez.
"Sistema de visión para el equipo de robots autónomos del ITAM"
Tesis ITAM, México, 2004

[3] Gonzalo Pajares, Jesús M. De la Cruz, José M. Molina, Juan Cuadrado y Alejandro López.
"Imágenes Digitales, Procesamiento práctico con Java"
Alfaomega Ra-Ma 2004

[4] Marcos Faúndez Zanuy
"Tratamiento digital de voz e imagen y aplicación a la multimedia"
Alfaomega Marcombo 2001

[5] MSDN Library
"Introduction to Direct Show"
DirectShow SDK Documentation, 2004

[6] González, Woods
"Digital Image Processing"
Adison-Wesley Publishing Company, 1993.

[7] Anil K. Jain
"*Fundamentals of Digital Image Processing*"
Prentice Hall 1989

[8] Yu Hen Hu, Jenq-Neng Hwang
"The Electrical Engineering and Applied Signal Pprocessing Series Handbook of Neural Network Signal Processing"
CRC PRESS

[9] Ernesto Bribiesca
"A new chain code"
Pattern Recognition 32 (1999) 235-251

[10] **http://www.cs.cmu.edu/~brettb/robocup/**

[11] **http://www.fotonostra.com**

[12] http://home.no.net/dmaurer/~dersch/barrel/barrel.html

[13] Alberto Leon-Garcia
"Probability and Random Processes for Electrical Engineering"
Addison-Wesley Publishing Company, 1994.

[14] James C. Wyant, Katherine Creath
"Basic Wavefront Aberration Theory for Optical Metrology"
Optical Sciences Center, University of Arizona and WYKO Corporation, Tucson,Arizona

[15] Vicente Carbonell, Marcelo Santalo
"Geometría Analítica"
Grupo Editorial Éxodo, 4ª Edición, 2004

[16] Hirsch, Schoen
"Trigonometría Conceptos y aplicaciones"
McGraw-Hill

[17] Horia Vlad Balan
"Visual Object Recognition in RoboCup"
Mayo 6, 2004

[18] James Bruce
"Realtime Machine Vision Perception and Prediction"
School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213-3891
Mayo 2000

[19] Bruce, Balch, Veloso
"Fast and inexpensive color image segmentation for interactive robots"
Proceedings of the 2000 IEEE/RSJ International Conference on Intelligent Robots and Systems. (IROS '00), 2000.

[20] Karpati,D´ Andrea Lee
"Real Time Visual Perception for Autonomous Robotic Soccer Agents"
Cornell University RoboCup Team: Big Red Bots, 1999.
http://robocup.mae.cornell.edu/documentation/robocup/1999/Vision_Manual.pdf

[21] Ketill Gunnarsson
"The Color and the Shape Automatic On-line Color Calibration for Autonomous Robots"
Fachbereich Mathematik und Informatik der Freien Universität Berlin,

Institut für Informatik, Takustraße 9, 14195 Berlin, 1 Noviembre 2005

[22] Tordoff and David W Murray
"The impact of radial distortion on the self-calibration of rotating cameras"
Department of Engineering Science, University of Oxford
Parks Road, Oxford OX1 3PJ, UK

[23] http://www.fu-fighters.de

[24] Mayer, G., Utz, H., Kraetzschmar, G.K.
" Towards autonomous visionself-calibration for soccer robots."
IEEE/RSJ International Conference on Intelligent Robots and Systems (2002) 214-219.

[25] D. Cameron, N. Barnes,
"Knowledge-Based Autonomous Dynamic Colour Calibration"
Robocup Symposium, 2003, Padua, Italy, July, 2003.
RoboCup 2003 award-winning paper: Engineering Challenge Award.

[26] Austermeier, H., Hartmann, G., Hilker, R.
"Color-calibration of a robot vision system using self-organizing feature maps."
Artificial Neural Networks - ICANN 96.
1996 International Conference Proceedings (1996) 257-62.

[27] G. Buchsbaum.
"A spartial processor model for object color perception."
In Journal of the Franklin Institute, Volume 310, Seiten 1-26. 1980.

[28] Slav Petrov.
" Computer Vision, Sensorfusion und Verhaltenssteuerung f¨ur Fussball-Roboter"
Master thesis (Diplomarbeit), Institut für Informatik, Freie Universität Berlin, 2004.

[29] http://www.hypermaths.org

[30] Maria Isabel Acosta Buitrago, Camilo Alfonso Zuluaga Muñoz
"Tutorial sobre Redes Neuronales Aplicadas en Ingeniería Eléctrica y su Implementación en un Sitio Web"
Universidad Tecnológica de Pereira, Facultad de Ingeniería Eléctrica, Octubre 2000

[31] http://www.correodelmaestro.com/anteriores/2004/febrero/1anteaula93.htm