UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

PROGRAMA DE MAESTRÍA Y DOCTORADO EN INGENIERÍA

FACULTAD DE INGENIERÍA

SISTEMA DE RECONOCIMIENTO DE PATRONES USANDO PROCESAMIENTO DE IMÁGENES PARA LOCALIZACIÓN DE ROBOTS MÓVILES

T E S I S

QUE PARA OPTAR POR EL GRADO DE:

MAESTRO EN INGENIERÍA

ELÉCTRICA-PROCESAMIENTO DIGITAL
DE SEÑALES E IMÁGENES

P R E S E N T A :

FRANCISCO JAVIER RODRÍGUEZ GARCÍA

TUTOR:
DR. JESÚS SAVAGE CARMONA

2007

JURADO ASIGNADO

Presidente:	DR. ESCALANTE RAMÍREZ BORIS
Secretario:	DRA. MEDINA GOMEZ LUCIA
Vocal:	DR. SAVAGE CARMONA JESUS
1er suplente:	DR. EN ING. PÉREZ ALCÁZAR PABLO ROBERTO
2do suplente:	DR. RIVERA RIVERA CARLOS

Lugar donde se realizó la tesis:

Laboratorio de Biorróbotica, 2do piso, Ed. de Posgrado de Ingeniería "Bernardo Quintana", Facultad de Ingeniería, UNAM

TUTOR DE TESIS

DR. JESÚS SAVAGE CARMONA

_____

Firma

Agradecimientos

A mi mamá y a toda mi familia por todo el apoyo que siempre me han brindado en todos mis proyectos, ya que sin su soporte hubiera sido muy difícil haber llegado hasta donde estoy ahora

Al Dr. Jesús Savage por haberme tenido la confianza de iniciar y llevar a cabo este proyecto, así como por todas los apoyos y oportunidades dados durante mi estancia en su laboratorio

A mis profesores por haberme brindado un poco de su tiempo y mucho de su conocimiento

A todos mis amigos por alentarme siempre en cada proyecto que inicio y por el apoyo que me dan hoy y siempre

A CONACYT por brindarme su apoyo económico, ya que de otra manera no habría sido posible concluir mis estudios de maestría

A la UNAM por otorgarme siempre de forma desinteresada una educación integral, completa y de calidad

A todas aquellas personas que de una u otra manera han contribuído en mi desarrollo personal, profesional y académico

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

1. INTRODUCCIÓN

1.1. Introducción

La robótica es un campo multidisciplinario que ha experimentado grandes avances desde su concepción, y ha sido utilizada de muy diversas maneras, desde líneas de producción totalmente automatizadas hasta productos de consumo diario como impresoras, además de ser adoptado tanto como una forma de entretenimiento como de exploración científica.

Un robot está formado por diversos elementos que interactúan entre sí, siendo cada uno de ellos un campo de estudio por sí mismo. En forma general un robot está compuesto por tres partes principales: los sensores de entrada, la inteligencia, y los actuadores. Los sensores son los elementos que le permiten al robot conocer la dinámica de su ambiente. La inteligencia es la parte que decide las acciones a tomar según la información entregada por los sensores. Finalmente, los actuadores se encargan de llevar a cabo lo que la inteligencia haya decidido, p.ej., moverse, detenerse, dar un paso, etc.

Una de las partes más críticas de un robot es el sistema de sensado, ya que de él dependen las acciones a tomar por parte del sistema de inteligencia, por lo que si la información que envían los sensores es errónea, o está corrompida por ruido, las decisiones tomadas también serán erróneas. Existen muchos tipos de sensores: de contacto, de luz, de movimiento, ultrasónicos, de imágenes, etc, y el tipo de sensor a utilizar depende en gran medida del propósito para el cual el robot ha sido diseñado y del ambiente en el cual éste operará.

Una de las formas de lograr avances en el campo de la robótica es a través de competencias que desafían el ingenio y la habilidad de los participantes. Este tipo de competencias se llevan a cabo en todas partes alrededor del mundo, siendo unas locales y otras globales, cuyos contendientes van desde aficionados hasta investigadores de renombre, siendo la mayoría estudiantes pre y post graduados. Una competencia muy popular en éste ámbito es la llamada Robocup, cuya primer edición fue organizada en Japón en 1997. La intención de esta competencia es lograr que equipos de robots jueguen fútbol sóccer, aunque con el paso del tiempo se han ido agregando otras categorías.

El Laboratorio de Biorrobótica del Posgrado de Ingeniería de la UNAM

decidió formar un equipo de estudiantes, dirigidos por el Dr. Jesús Savage Carmona, para participar en dicha competencia. Los estudiantes reunidos pertenecen a diversos campos del conocimiento: electrónica, procesamiento de imágenes, inteligencia artificial y mecánica. Por lo que a este trabajo se refiere ha sido el área de procesamiento de imágenes cuyo estudio concierne.

Dada la naturaleza de la competencia existen dos formas de sensar el ambiente de operación de los robots: cámaras de video globales, y cámaras de video a bordo. Las primeras son colocadas por encima del campo de juego, por lo que una o dos cámaras son suficientes para abarcar toda el área de interés. Para las cámaras de video a bordo se requiere una cámara montada en cada robot. Además los robots utilizan parches en su tapa superior, para la de identificación y orientación de los mismos.

El presente trabajo utiliza visión global y tres parches (más uno obligatorio que sirve para identificar a cada equipo durante la competencia) para lograr la detección y orientación de cada uno de los robots de nuestro equipo. El sistema de visión desarrollado detecta y procesa la información que envía la cámara global y a continuación envía (vía sokets de internet) las coordenadas y número de identificación de cada robot al sistema de inteligencia artificial, quien diseña las estrategias de juego y transmite de forma inalámbrica las instrucciones de movimiento al robot.

El resto del capítulo explora en forma más detallada el presente proyecto.

1.2. Robocup

La Robocup es una competencia paralela a la competencia mundial de fútbol entre humanos. La diferencia es que la Robocup es jugada por robots en lugar de humanos, y es con fines educativos. Su principal objetivo es promover la investigación en robótica e inteligencia artificial (AI) probando e integrando soluciones en una amplia variedad de tecnologías. El lema de esta liga es [1]:

> Desarrollar un equipo de robots humanoides totalmente autónomos
> que puedan ganarle al campeón del mundo de fútbol en el año 2050

Esto presenta una serie de retos que deben ser salvados por los diseñadores y programadores en diversas disciplinas como mecánica, AI, procesamiento de imágenes, programación, etc.

1.3. Robótica

Proyectos de esta magnitud sólo son posibles con un grupo multidisciplinario, ya que se requieren especialistas, o al menos entusiastas, en cada área de trabajo, y aunque es posible que un único individuo pueda llegar a obtener buenos resultados, la mayoría de los equipos alrededor del mundo están formados por grupos dedicados a un área en particular.

Para el proyecto que ocupa al presente trabajo se cuenta con personal específicamente involucrado en cada una de las áreas que intervienen en el proyecto: procesamiento de imágenes, electrónica, inteligencia artificial, comunicaciones, control y mecánica.

La forma de trabajo se basó en una cadena de lazo cerrado donde cada grupo tiene bien definido lo que le corresponde hacer: tomar los datos del nivel anterior, procesarlos y enviarlos al nivel siguiente. Es de lazo cerrado, porque, como muestra en la Fig(1.1), existe un canal de retroalimentación del movimiento real del robot (dado por la cámara) y el movimiento deseado establecido por el módulo de intelgencia artificial.

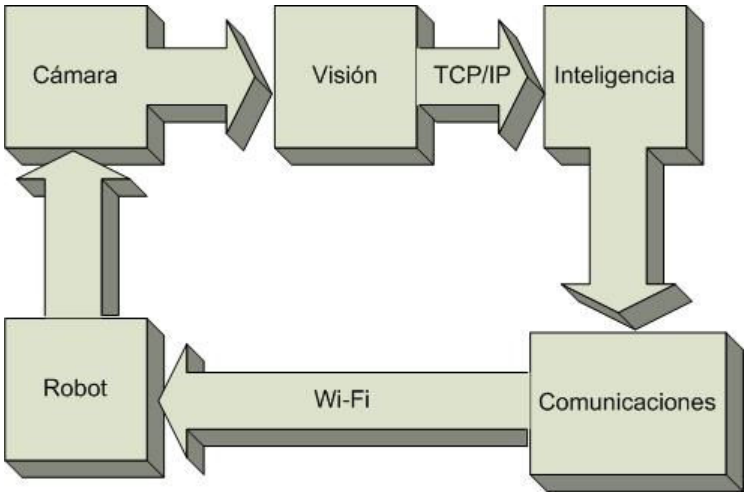

Fig. 1.1: Lazo cerrado del sistema completo de competencia Robocup. Las flechas con texto dentro indican el tipo de comunicación entre los subsistemas

1.4. Visión

Un reto fundamental es la percepción de los robots del mundo que los rodea. Un robot que no cuente con algún tipo de sensado del exterior prácticamente

sería inútil. Dependiendo de la actividad a desarrollar el robot pudiera contar con algún tipo, o combinación, de sensores tal como infrarrojos, de toque, de temperatura, etc. Para el tema del presente trabajo el sensor es una cámara web con sensor CMOS.

Todo sería muy simple si sólo se tratara de colocar la cámara y alimentar imágenes en bruto; sin embargo, el módulo de inteligencia (el cual se encarga de generar las estrategias de juego) requiere de información fidedigna y en tiempo real de todo aquello que sucede en el campo de juego. Por lo tanto, el módulo de visión debe alimentar a dicho módulo con las posiciones de todos y cada uno de los robots, así como parámetros tales como velocidad, orientación y el movimiento de la pelota, y todo ello en tiempo real. Esto no es una tarea trivial ya que se necesita hacer un rastreo de la imagen en bruto, procesarla, identificar robots y pelota, obtener sus parámetros, y finalmente enviarla al módulo de inteligencia.

Para que el módulo de visión logre esta tarea se particionó en 7 submódulos:

1. Captura
2. Preprocesamiento
3. Calibración
4. Segmentación
5. Regiones
6. Identificación
7. Comunicaciones

Cada uno de estos submódulos será tratado en detalle en el siguiente capítulo.

1.5. Trabajos anteriores

En [2] se tiene un trabajo completo de visión para la Robocup en la Small League desarrollado en el Instituto Tecnológico Autónomo de México (ITAM), donde describen su sistema de visión y además ponen al alcance del público sus códigos fuente, además de la información proporcionada en la propia tesis. En Carnegie Mellon University [3] existe toda una serie de trabajos que documentan las actividades desarrolladas por muchos años y que los han llevado tanto a convertirse en uno de los mejores equipos del mundo, así como desarrollar

una excelente infrastructura en las diversas ramas de la robótica. También en [6] se pueden encontrar los trabajos publicados por Freie Universität Berlin, quienes son los actuales campeones del mundo en 2004 en la ya mencionada categoría.

Un punto importante a recalcar es que las universidades mencionadas, dentro de su infraestructura, cuentan con cámaras digitales de alta resolución, lo que les permite salvar algunos obstáculos, p. ej. imágenes ruidosas, al momento de hacer su procesamiento. Aunque este planteamiento es sin duda el más práctico, no siempre se cuenta con tal equipo, ya que además de las cámaras se requieren tarjetas de captura de video digital, lo que encarece al proyecto y desmotiva a equipos con pocos recursos. Debido a ello, el sistema de visión desarrollado hace uso de webcams para el puerto USB, lo que permitirá que no sólo instituciones con recursos limitados, sino también entusiastas de la robótica puedan iniciar sus propios proyectos.

1.6. Objetivos Generales

Los objetivos que se desean alcanzar al término de este proyecto son

1. **Desarrollar un sistema de visión robusto que sea capaz de detectar a todos y cada uno de los robots en ambos equipos, así como a la pelota**
2. **Entregar estos datos a través de sockets de internet a la máquina encargada de la inteligencia y estrategia**

1.7. Objetivos específicos

Los objetivos específicos son

1. Desarrollar una interfaz gráfica amigable
2. Desarrollar un sistema de calibración semiautomático de colores
3. Desarrollar librerías que puedan ser usadas en otros proyectos

1.8. Organización del trabajo

- En el capítulo 2 se exploran las definiciones y los fundamentos teóricos necesarios para el desarrollo del proyecto

- En el capítulo 3 se implementan las técnicas y algoritmos requeridos para lograr los objetivos planteados
- En el capítulo 4 se tienen las pruebas y resultados obtenidos luego de la implementación
- En el capítulo 5 se tienen las concluisiones a las que se llegó, así como a lo que pueda quedar pendiente

2. MARCO TEÓRICO

2.1. Robocup categoría Small Size

La Robocup fue diseñada para realizar avances en robótica y AI a través de juegos amistosos. La categoría SmallSize es una de las divisiones de tal competencia. También conocida como F180, esta categoría se enfoca en el problema de la cooperación inteligente multi-agentes y control en un ambiente altamente dinámico con sistemas híbridos centralizados/distribuídos.

Un juego en esta categoría requiere de dos equipos de cinco robots cada uno. Cada robot debe cumplir con las dimensiones dadas en las reglas de la F180: el robot deberá estar contenido en dentro de un círculo de 180 mm de diámetro, con una altura de 150 mm, excepto que se use visión a bordo. El partido se lleva a cabo en un campo verde de 2.8 m de largo por 2.3 m de ancho con una bola de golf naranja como balón. Hay dos tipos de robots, con visión a bordo y con visión global. Estos últimos son los más utilizados y usan una cámara que está por encima del campo (fijada a un riel a 3 m), así como una computadora fuera de este para identificar y rastrear a los robots dentro del campo. En la visión local los robots incorporan los sensores dentro de ellos mismos. La información adquirida puede ser procesada en el mismo robot o enviada a una computadora fuera del campo. También se tiene una computadora que realiza las veces de árbitro. Usualmente las computadoras que están fuera del campo son las que realizan casi todo el trabajo de procesamiento necesario para la coordinación y movimiento de los robots. Las comunicaciones son inalámbricas. En la Fig(2.1) se puede apreciar a tres robots del equipo small-size del Laboratorio de Biorobótica del Edificio de Posgrado de la Facultad de Ingeniería de la UNAM

2.2. Robots

La palabra *robot* deriva de la palabra checa *robota*, que significa **trabajo forzado**, se hizo popular cuando la obra de teatro de Karel Capek's *Rossum's Universal Robot* fue interpretada en Francia en los años 20's. En esta obra

Fig. 2.1: Equipo de robots del Laboratorio de Biorobótica

pequeñas criaturas artificiales y antropomórficas obedecían estrictamente las órdenes de su dueño. Hoy en día la robótica puede ser definida, según [5], como la teoría y la práctica de la automatización de tareas, que dada su naturaleza, eran antiguamente reservadas para los humanos. Así p. ej. una impresora pueden considerarse como un robot. Otra definición es : Un robot es un dispositivo generalmente mecánico que desempeña tareas automáticamente, ya sea de acuerdo a supervisión humana directa, a través de un programa predefinido o siguiendo un conjunto de reglas generales, utilizando técnicas de inteligencia artificial. Generalmente estas tareas reemplazan, asemejan o extienden el trabajo humano, como emsamble en manufactura, manipulación de objetos pesados o peligrosos, trabajo en el espacio, etc. [11]

Siguiendo estas definiciones se pueden identificar tres líneas de investigación alrededor de estos dispositivos

1. Robots individuales, situados en una localización fija o móvil (robots móviles)

2. Robots trabajando en conjunto con otros robots o máquinas
3. Robots que son controlados a distancia

Así mismo se desprenden dos características importantes

- *Versatilidad*: La versatilidad de un robot depende de su geometría y su capacidad mecánica
- *Autoadaptación al ambiente*: Se refiere al potencial que debe tener el robot para adaptarse a situaciones que no fueron completamente definidas en un principio. Para esto el robot utiliza
  - su habilidad para percibir el ambiente (por el uso de sus sensores)
  - su capacidad para el análisis espacio-objetivo y ejecutar un plan de operación
  - sus modos de comandos automáticos

2.2.1. Estructura general de un robot

Se pueden identificar cuatro partes interactivas en un robot

1. *El dispositivo equipado con actuadores*. Es el tipo de máquina diseñado para llevar a cabo una tarea. Las diferentes articulaciones son controladas por actuadores, generalmente eléctricos o pneumáticos.
2. *El ambiente*. Se refiere a todo aquella que se encuentra alrededor del robot en su ambiente de trabajo
3. *La tarea, u objetivo*. Una tarea se define como la diferencia entreo dos estados de un ambiente –el estado inicial y el estado final luego de haber completado la tarea.
4. *El cerebro del robot*. Es la parte del robot que genera las señales acorde a información *a priori*, e información *a posteriori* dada por el conocimiento de los estados presente y pasado del ambiente y del robot mismo.

Actualmente la robótica es un amplio campo de estudio que abarca varias disciplinas de la ciencia y la técnica, incluyendo cinemática, dinámica, planeación, control, sensado, lenguajes de programación e inteligencia artificial, entre otras [6].

- *Cinemática y dinámica.* La *cinemática* estudia la descripción analítica del desplazamiento espacial del robot como una función del tiempo, mientras que la *dinámica* estudia las formulaciones matemáticas del movimiento de éste.

- *Planeación y control de movimiento.* La *planeación* se refiere a interpolar y/o aproximar la trayectoria que el robot debe seguir para llegar a su objetivo con una clase de funciones polinomiales y generar, por lo tanto, un vector de puntos de control que el robot utilizará para llegar a su destino. Por otro lado se tiene que el *control de movimiento* consiste de (1) obtener el modelo dinámico del robot, y (2) usar este modelo para determinar las estrategias que logren la respuesta y el desempeño deseados.

- *Sensado.* El uso de sensores externos le permiten al robot interactuar con su medio ambiente de forma flexible. La función de estos se puede dividir en *estados internos* y *estados externos*. Los estados internos son la detección de las variables de que el robot usa, p. ej. su propia posición, mientras que los estados externos tratan con la detección de las variables del ambiente, p. ej. proximidad, temperatura, etc.

- *Lenguajes de programación.* Es la forma de abordar el problema de la comunicación con un robot. Los programas, o secuencias de intrucciones, describen las operaciones que el robot debe realizar, permiten que éste lleve a cabo diferentes tareas con sólo ejecutar el programa adecuado. Es la programación la que ha dado a los robots su flexibilidad y versatilidad.

- *Inteligencia artificial.* Es una ciencia que intenta la creación de programas para máquinas que imiten el comportamiento y la comprensión humana, que sea capaz de aprender, reconocer y pensar. De esta manera es posible que el robot aprenda de su ambiente y tome sus propias decisiones, inclusive en situaciones para las cuales no fue programado. Este módulo es el que se encarga de buscar la mejor trayectoria para que el robot llegue a su destino.

2.3. Reconocimiento de patrones

Así como en los humanos, la capacidad de ver provee al robot de un sofisticado mecanismo de sensado que le permite responder a su ambiente en una forma inteligente y flexible, por lo que ésta se puede considerar como la mejor fuente de sensado en un robot. Pero como es de esperarse, el procesamiento

de las imágenes capturadas es considerablemente más complejo que con otro tipo de sensores.

La visión robótica se puede definir como el proceso de extraer, caracterizar e interpretar la información en imágenes de un mundo tridimensional [6]. Este punto de vista es muy diferente al de procesamiento de imágenes, en el cual se estudian las tranformaciones de imagen a imagen, no las construcciones básicas dentro de éstas. Las descripciones son el prerrequisito básico para reconocer, manipular y pensar en objetos [7].

El reconocimiento de patrones, también conocido como visión de máquinas o de computadoras, se puede dividir en seis áreas principales

2.3.1. Sensado o captura

Es el proceso por el cual se obtiene una imagen. y el cual se encarga de la interacción entre el hardware y la aplicación. Ya que cada fabricante de cámaras, y en general de cualquier dispositivo periférico, tiene sus propias reglas y protocolos para utilizar sus productos, es necesario contar con una interfaz que unifique los diferentes criterios y que de alguna manera haga que el programador que va utilizar las imágenes se deslinde de tan engorrosa tarea. Para ello se utilizó DirectX, el cual es un estándar de programación de video bajo el sistema operativo Windows. Esta interfaz se hace cargo de todo aquello detrás de la escena y sólo entrega una matriz de video independiente de la fuente que lo genera.

2.3.2. Preprocesamiento

Son las técnicas que se encargan de mejorar la imagen en bruto que le llega del submódulo anterior. La gran mayoría de las veces no se cuenta con una iluminación constante, o los sensores de las cámaras no son lo bastante buenos, entre otras cosas, lo que degenera a la imagen y la hace difícil de utilizar por los siguientes submódulos. Por lo tanto, la tarea principal del preprocesamiento es la de tratar de corregir esos defectos. Así mismo, otra función de este submódulo es la de cambiar entre diferentes representaciones del color, p. ej. de RGB a HSI, que como se verá más adelante, es piedra angular del sistema de visión.

2.3.3. Segmentación

Se encarga de aislar los objetos de interés (en este caso pixeles) de la escena, es decir, cada objeto que se encuentre se aislará del resto de la imagen. Este proceso agrupa pixel por pixel siguiendo lineamientos establecidos que

se verán más adelante. Este proceso se apoya del submódulo de calibración, el cual se encarga de definir las clases (o regiones) para la detección de los colores. Gran parte del buen funcionamiento del sistema de visión recae en este submódulo, ya que si no se están detectando adecuadamente los colores no se podrá reconocer a los objetos en escena. Aunque existen diversos esquemas (determinísticos y probabilísticos) se tiene el compromiso de la precisión de la detección contra la velocidad de procesamiento, ya que este módulo trabaja en tiempo real, por lo que habrá que decidir el esquema que mejor se adapte a estas necesidades en particular. Para ello el módulo divide en clases o regiones. Una clase, o patrón, es una región delimitada en base a una función discriminante. Se tendrán tantas clases como colores se deseen detectar. Es un cuantizador que opera sobre vectores de colores.

2.3.4. Descripción

Es el cálculo de las características que diferencían un objeto de otro. p. ej. tamaño, forma, etc. Se encarga de la agrupación de los objetos y de darles nombre. Es decir, del paso anterior ya se tienen objetos aislados (pixeles), entonces este se encarga de juntarlos (siguiendo también ciertos lineamientos) para comenzar a identificarlos. P. ej. ¿se trata de la pelota?, ¿es un parche de color azul?. El resultado de este submódulo es una regla de decisión que asigna cada objeto a una clase particular (al color al cual pertenece la región que se ha creado), y a partir de ahí decidir si es un parche válido o la pelota.

2.3.5. Reconocimiento

Es la identificación de los objetos dentro de la escena, p. ej. otros robots, muros, etc. Agrupa los objetos y encuentra en la escena entes menos abstractos, como robots. Se puede decir que este es la capa de más alto nivel, ya que es la conclusión del proceso de búsqueda y clasificación de objetos concretos en la escena. En este nivel se identifica a cada robot propio, así como algunas de sus características más sobresalientes, como son la identifiación del robot, su posición, velocidad y ángulo. De los robots pertenecientes al otro equipo sólo se determina su posición, ya que no es posible saber de antemano qué sistema de identificación utilizarán, además de que sería impráctico tratar de abarcar todos los sistemas posibles.

2.3.6. Interpretación

Da un significado al ensamble de objetos reconocidos, es decir, establece si un objeto hayado se corresponde con un objeto válido como un robot o la

pelota.

2.3.7. Comunicaciones

Se encarga de transmitir a un sistema remoto la información recabada del paso anterior. Aunque estrictamente no tiene nada que ver con la visión y el reconocimiento, es importante mencionarlo ya que el módulo que utilizará estos datos probablemente se encuentre en otra máquina. Dado que los algoritmos de Inteligencia Artificial (AI) pueden ocupar mucho tiempo de proceso de CPU, se decidió separar los módulos de visión y de AI, por lo que es es necesario contar con un sistema fiable de comunicaciones entre computadores. El sistema elegido fue el de sockets de internet, gracias a que son fiables y fáciles de usar, además de que son independientes de la plataforma y casi todos los sistemas operativos los soportan.

En la Fig(2.2) se puede apreciar esta secuencia gráficamente.

A su vez, es conveniente agrupar estas divisiones acorde a su grado de sofisticación, p.ej.

- *Bajo nivel*. Son los procesos primitivos en el sentido que se consideran "de reacción automática" y que no requieren de ningún tipo de inteligencia de parte del sistema de visión, y para fines de este trabajo el área de *sensado*, de *preprocesamiento* y de *conversión de espacios de color* se tratarán como de bajo nivel.

- *Nivel intermedio*. Son los procesos que extraen, caracterizan y etiquetan componentes en una imagen resultado del proceso de bajo nivel. Las áreas de *segmentación*, *descripción* y *reconocimiento* serán consideradas como de nivel intermedio.

- *Alto nivel*. Son los procesos que intentan emular el razonamiento humano. El área de *interpretación* se considerará de este tipo.

2.4. Espacios de color

Para distinguir de manera precisa a los objetos basados en su color es necesario escoger algún sistema de representación de colores, llamado *espacio de color*.

Un espacio de color es una especificación de un sistema coordenado y un subespacio dentro de él donde cada color se representa por un punto. El propósito de un espacio de color es la de facilitar la forma en que se especifican los colores en forma estandarizada [14].

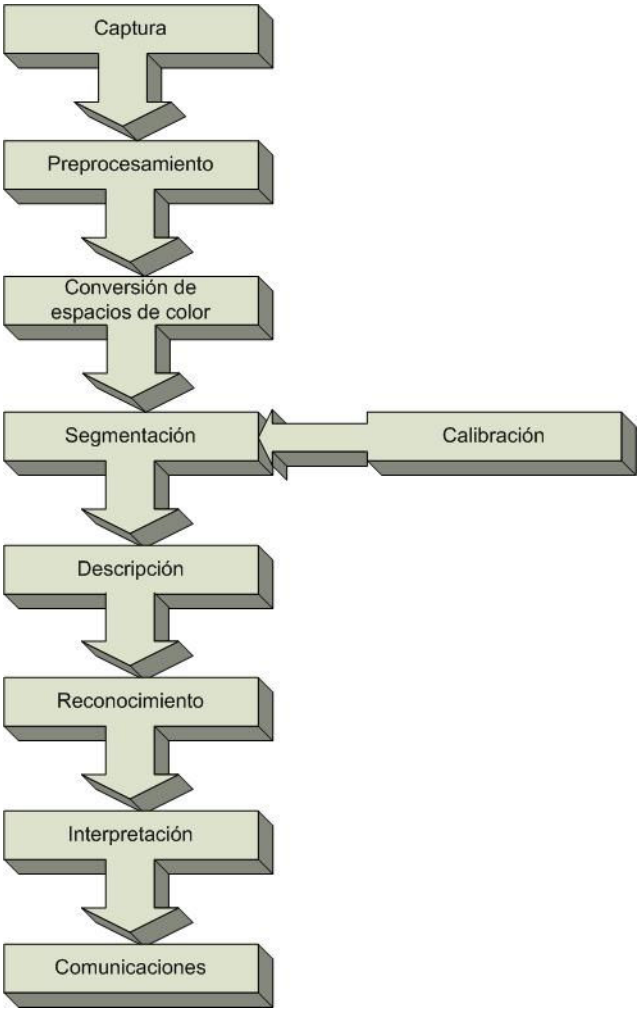

Fig. 2.2: Secuencia de pasos para el reconocimiento de los objetos en escena

Muchos de los espacios de color actuales están orientados ya sea hacia el hardware (p. ej. monitores e impresoras) o hacia aplicaciones donde el objetivo principal es la manipulación de colores (p. ej. animaciones). Los espacios de color más utilizados de cara al hardware son el RGB (red, green, blue), el cual es utilizado en monitores y cámaras; el CMY (cyan, magenta, yellow) y el CMYK (cyan, magenta, yellow, black) para impresión; y el HSI (hue, saturation, intensity) el cual es el que más se corresponde con la manera en que el humano describe e interpreta el color [14].

2.4.1. Espacio de color RGB

La longitud de onda que los ojos pueden detectar es solamente una pequeña parte del espectro de energía electromagnético; es a esto a lo que se le llama *espectro de luz visible*. En un extremo de este espectro se encuentran las ondas más cortas de luz que se perciben como azul. En el otro extremo se tienen las longitudes de onda más grandes que se perciben como rojo. Los otros colores que se perciben en la naturaleza se encuentran en algún lugar entre el rojo y el azul. Más allá de estos límites se tienen las ondas cortas de la luz ultravioleta y los rayos X, y las ondas más largas de infrarrojo y de radio, las cuales son percibidas por el ojo humano.

Diviediendo al espectro visible en tres los colores predominantes son el rojo, verde y azul, los cuales son considerados los tres colores primarios, de donde se desprenden todos los demás colores.

El sistema aditivo de colores, o RGB, por su siglas en inglés, tiene que ver con la luz emitida por la fuente, antes de que el objeto refleje la luz. Este proceso de reproducción aditiva mezcla cierta cantidad de luz roja, verde y azul para producir otros colores. Cuando se combinan los tres en la misma cantidad se obtiene el blanco.

Todos los dispositivos de captura de imágenes utilizan el sistema RGB para recolectar la información necesaria para reproducir una imagen en color. Entre estos dispositivos se tienen las cámaras digitales, las webcams, escáners de cama, viedocámaras, entre otros. Esta es una representación natural; por un lado las cámaras en color descomponen cada pixel de la imagen en estas tres componentes, y por el otro un monitor tiene tres cañones de electrones, uno para cada color. La digitalización de la imagen transforma ésta de una representación analógica a su correspondiente representación en números (bits y bytes) que una máquina puede procesar. Al número de bits utilizados para representar cada pixel de la imagen se le conoce como *profundidad de pixel*. Considerando que a cada color se le den 8 bits, se dice que su profundidad de color es de 24 bits, y en 24 bits se tienen $(2^8)^3 = 16,777,216$ colores. En la Fig(2.3) se puede apreciar el cubo RGB. Sin embargo, esta representación no es la más adecuada para la detección de colores ya que los cambios en la iluminación generalmente afectan a las 3 componentes, lo cual no es deseable.

Cada punto de color en una imagen se guarda en memoria como la combinación de valores de los tres colores básicos (en el espacio RGB). A esta combinación se le pueden aplicar diversos procedimientos para alterar la forma en que percibimos la imagen. De forma particular tenemos cuatro posibilidades: a) equilibirio de color, donde se realza o se disminuye un color en especial; b) brillo, donde se aclara u obscurece una imagen; c) contraste, donde se acentúan

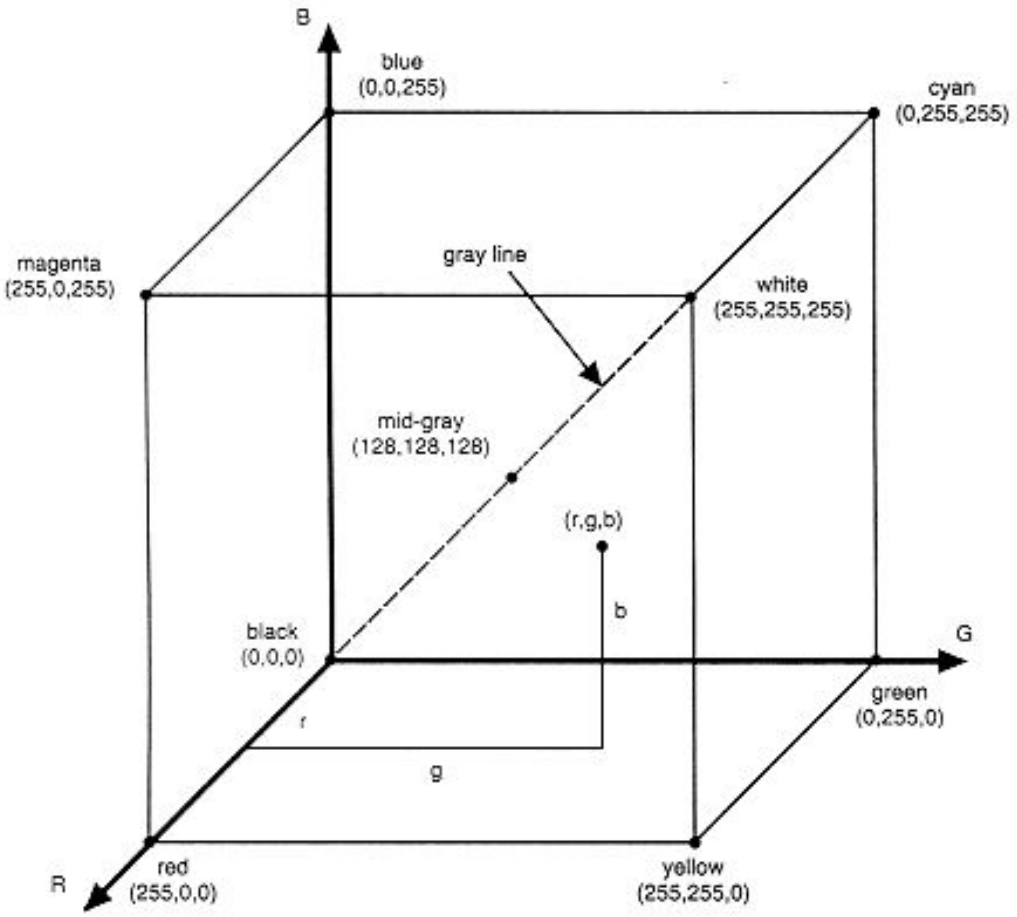

Fig. 2.3: Cubo RGB

las diferencias entre los colores; y d) saturación, es decir, colores vivos o grises.

Estos procedimientos son previos a la conversión al espacio HSI.

2.4.2. Equilibrio de color

Se refiere a la mayor o menor importancia cuantitativa que se le da a un color u otro en la totalidad de la imagen, es decir, algunos colores pesarán más que otros.

El algoritmo consiste en añadir o restar una constante a los valores RGB de todos los pixeles individuales [12]. P. ej. si se quisiera dar una tonalidad más azulada a una imagen $M_{RGB}(x, y)$ se le suma una cantidad constante K_B a cada pixel; o por el contrario, se le puede sustraer una cantidad para hacerla menos azul. Si v se refiere a cualesquiera de las componentes RGB entonces se

tiene que

$$M_{RGB}(x, y) = M_v(x, y) + K_v \tag{2.1}$$

procurando que $0 \leq M_{RGB}(x, y) < 255$, ya que de lo contrario pueden aparecer errores inesperados, debido a que cada componente se almacena en una palabra de 8 bits, entonces se tiene que la cuenta máxima es de $2^8 = 256$. Esta misma situación aplica para todas aquellas operaciones que involucren la alteración de las componentes RGB.

2.4.3. Brillo

Este proceso es equivalente a la manipulación del control de brillo de un televisor o monitor. Para aclarar una imagen obscura se aumenta el brillo; para obscurecer una imagen clara se disminuye. Este algoritmo es un caso especial de aquella dada por la Ec(2.1) en el que se aplicará el mismo incremento, o decremento, a las tres componentes de color.

$$M_{RGB}(x, y) = [M_R(x, y) + K_R] + [M_G(x, y) + K_G] + [M_B(x, y) + K_B] \tag{2.2}$$

2.4.4. Contraste

Esta operación es más compleja que las anteriores. *Contraste* signinfica "diferencia notable"; por lo tanto aumentar el contraste será acentuar las diferencias entre los colores, mientras que reducir el contraste será disminuirlas.

La mínima diferencia se produce en el gris cuando rojo, verde y azul tienen el mismo valor; pero además se debe elegir un gris que esté a medio camino entre la total luminosidad y la total obscuridad. Este "gris neutro", el de mínimo contraste, puede conseguirse asignando al rojo, verde y azul el valor 255/2, redondeando, 127, por lo tanto

$$Gn = RGB(127, 127, 127) \tag{2.3}$$

Por otra parte, la máxima diferencia se da entre los colores puros (rojo, verde y azul) y entre estos y sus complementarios (amarillo, cyan, violeta). Para aumentar el contraste de una imagen debe usarse un algoritmo que aumente las diferencias entre los colores, conduciéndolos hacia los colores puros; y, para reducir el contraste, hay que conseguir que todos los colores tiendan hacia el gris neutro. Para ello se multiplican los valores por una constante (factor de constraste). Sin embargo, antes de ello es necesario centrar el color, es decir, substraer 127 a cada una de las componentes del color, ya que de otra

manera sólo se lograría aclarar u obscurecer la imagen. Una vez que el color se centró, cada vez que una componente se multiplique por un factor de contraste mayor que uno, los valores negativos se volverán más negativos, y los positivos más positivos, alejándose todos del gris neutro (mayor diferencia entre ellos). Después de haber multiplicado por el factor de constraste se sumará de nuevo 127 a cada componente para que todos los valores sean positivos. De aquí se deduce que los valores mínimos y máximos para el factor de contraste serán 0 y 128, respectivamente. De 0 a 1 se reduce el contraste; de 1 a 128 se aumenta el constraste.

2.4.5. Saturación

La saturación se define como la tendencia de una imagen a tener colores vivos (muy saturados) o agrisados (poco saturados). Para cada pixel se calcula el valor de saturación como la diferencia entre los colores reales del pixel y el valor de gris que le correspondería si la imagen estuviera en tonos de grises. El valor de gris se calcula como el promedio de las 3 componentes:

$$M_{Gray}(x,y) = \frac{M_R(x,y) + M_G(x,y) + M_B(x,y)}{3} \tag{2.4}$$

Una vez calculada el valor de gris teórico, la saturación para cada uno de los tres colores será

$$\begin{aligned} Sat_R(x,y) &= R(x,y) - M_{Gray}(x,y) \\ Sat_G(x,y) &= G(x,y) - M_{Gray}(x,y) \\ Sat_B(x,y) &= Br(x,y) - M_{Gray}(x,y) \end{aligned} \tag{2.5}$$

Finalmente se multiplican estos valores por el incremento/decremento, K_R, K_G, K_B de saturación deseado:

$$\begin{aligned} M_R(x,y) &= M_R(x,y) + K_R[M_R(x,y) - Sat_R(x,y)] \\ M_G(x,y) &= M_G(x,y) + K_G[M_G(x,y) - Sat_G(x,y)] \\ M_B(x,y) &= M_B(x,y) + K_B[M_B(x,y) - Sat_B(x,y)] \end{aligned} \tag{2.6}$$

2.4.6. Espacio de color HSI

La necesidad del uso de un espacio de color alternativo al RGB fue porque en un primer intento dicho espacio no dió buenos resultados debido primordialmente a que la variación en la intensidad luminosa del ambiente afecta a las tres componentes, lo que hace muy díficil su calibración. Aunque no fue

cuantificado el error dado por el espacio RGB, las siguientes figuras demuestran la variabilidad de las componentes ante la modificación de la intensidad luminosa. En las figuras aparecen el mismo color para los dos espacios, RGB y HSI. En la figura de la izquierda se aprecian los valores para un color en particular, mientras que en la Fig de la derecha aparece el mismo color, pero menos luminoso, es decir, con menos luz, y puede apreciarse la forma en que las componentes RGB se movieron, en comparación con las componentes HSI, donde solamente la componente I varió. De aquí que haya sido necesario migrar del espacio RGB al espacio HSI.

Aunque existen otros espacios, p. ej. YUV, y variantes del mismo HSI, se decidió utilizar dado que este espacio de color refleja la manera que el humano percibe el color [14], como lo muestra la Fig(2.4).

En este formato, la H significa Hue(tinte), la S es Saturación, y la I es Intensidad [8]. Hue es el atributo de color que describe al color puro, mientras que la saturación da una medida del grado de cómo el color puro se diluye por la luz blanca, hasta alcanzar el blanco (cero saturación). La intensidad, también llamada *brillo*, es un descriptor subjetivo prácticamente imposible de medir. Este conlleva la noción acromática de intensidad y es uno de los factores clave para describir la sensación de color. Se sabe que la intensidad (en escala de grises) es el descriptor más utilizado en imágenes monocromáticas, por lo que sí es posible medir esta cantidad y darle una interpretación. Para los propósitos de este trabajo, esta cantidad será calculada como el promedio de los canales RGB de la imagen en color, aunque se llegan a utilizar otros esquemas con diferente ponderación. El modelo HSI desacopla la componente de intensidad de la portadora de información de color (tinte y saturación) en una imagen de color. Como resultado se tiene que el modelo HSI es una herramienta ideal para desarrollar algoritmos de procesamiento de imágenes basados en descripciones de color que son naturales e intuitivos a los humanos, ya que diferentes condiciones de iluminación tienen efectos mayores en la componente de intensidad I, pero muy poco efecto en el matiz H y en la saturación S, por lo que se puede restringir la atención a estas dos. Las Ec(2.7,2.8 y 2.9) muestran el algoritmo de conversión

$$H(x,y) = \cos^{-1}\left[\frac{(M_R - M_G) + (M_R - M_B)}{2\sqrt{(M_R - M_G)^2 + (M_R - M_B)^2 + (M_G - M_B)^2}}\right] \tag{2.7}$$

si $M_B > M_G \Rightarrow H(x,y) = 360 - H(x,y)$

$$S = 1 - \frac{3}{(M_R + M_G + M_B)}\,\text{mín}(M_R, M_G, M_B) \tag{2.8}$$

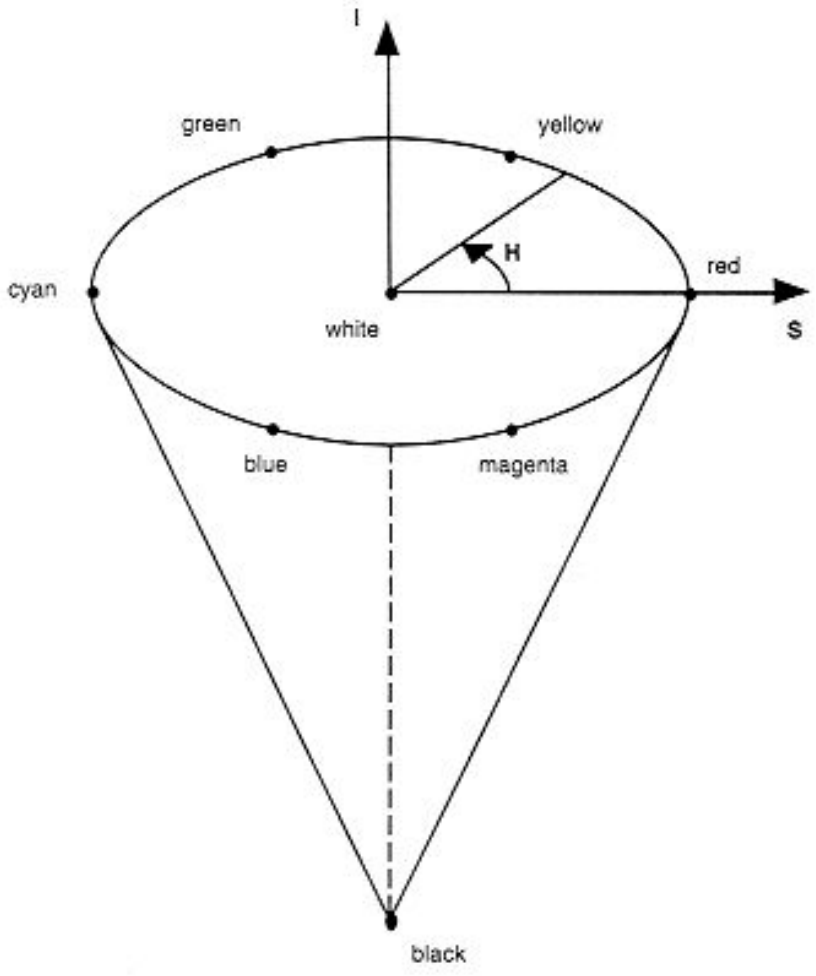

Fig. 2.4: Cono HSI

$$I(x,y) = \frac{M_R + M_G + M_B}{3} \tag{2.9}$$

2.4.7. Densidad de probabilidad e histograma

Son muy diversas las aplicaciones que se le pueden dar al histograma, y en este trabajo se utilizará para indicarle a la aplicación el rango que abarcará cada color para realizar posteriormente su discriminación. En total se tendrán tres histogramas (una para cada componente) y a través de una función de discriminación que involucra la media y la varianza de cada una de las tres componentes, se determinará a qué clase pertenece el pixel.

Se utiliza el histograma como alternativa a la función de densidad de probabilidad, ya que esta última opera sobre variables aleatorias continuas, mientras que las imágenes son variables aleatorias discretas.

Función de distribución acumulativa. La función de distribución acumulativa (CDF) de una variable aleatoria X se define como la probabilidad del evento $\{X \leq x\}$

$$F_x(x) = P[X \leq x] \tag{2.10}$$

para $-\infty < x < \infty$, es decir, es la probabilidad de que la variable aleatoria X tome un valor que esté en el rango $(-\infty, x]$, como se ve en la Fig(2.5)

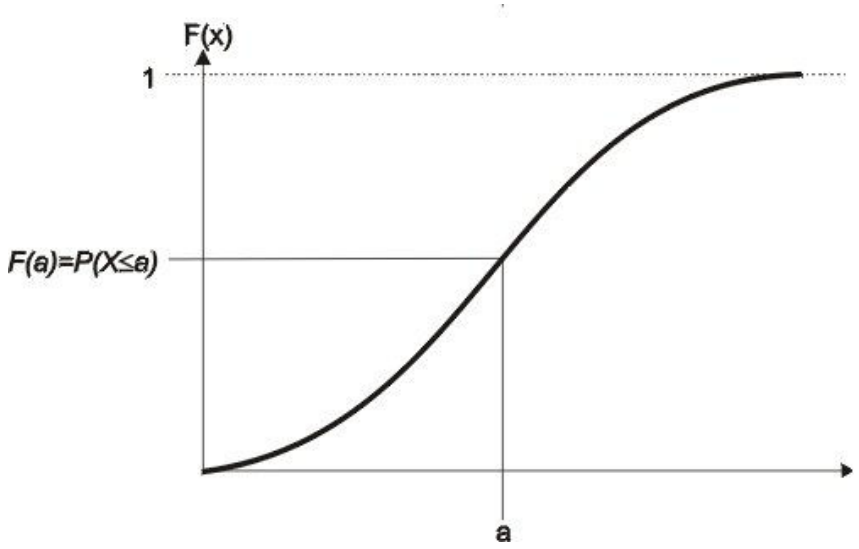

Fig. 2.5: Representación gráfica de la CDF

La CDF es una manera conveniente de especificar la probabilidad de todos los intervalos semi-infinitos en el eje de los reales de la forma $(-\infty, x]$ con las siguientes propiedades [13]

$$\begin{aligned} &0 \leq F_x(x) \leq 1 \\ &\lim_{x\to\infty} F_x(x) = 1 \\ &\lim_{x\to-\infty} F_x(x) = 0 \\ &F_x(a) \leq F_x(b); a < b \\ &F_x(b) = \lim_{h\to 0} F_x(b+h) = F_x(b^+) \end{aligned} \tag{2.11}$$

Función de densidad de probabilidad. La función de densidad de probabilidad (PDF), si existe, está definida como [13]

$$f_x(x) = \frac{dF_x(x)}{dx} \tag{2.12}$$

La PDF representa la densidad de probabilidad en un punto x de tal forma que la probabilidad de que X se encuentre en un pequeño intervalo en la vecindad de x, es decir, $\{x < X \leq x + h\}$, es

$$P\left[x < X \leq x + h\right] = F_x(x + h) - F_x(x) = \frac{F_x(x + h) - F_x(x)}{h} h \tag{2.13}$$

Si la CDF tiene derivada en x entonces h se hace muy pequeña

$$P\left[x < X \leq x + h\right] = F_x(x)h \tag{2.14}$$

Por lo tanto $f_x(x)$ representa la densidad de probabilida en el punto x en el sentido de que la probabilidad de que X se encuentre en un intervalo pequeño en la vecindad de x es aproximadamente $f_x(x)h$, ver Fig(2.6). La derivada de la CDF, cuando existe, es positiva puesto que la CDF es una función no decreciente de x.

Dado que todos los eventos que involucran a la variable aleatoria X pueden escribirse en términos de la CDF se tiene que dichas probabilidades pueden escribirse en términos de la PDF, por lo tanto se puede establecer que la PDF describe completamente el comportamiento de variables aleatorias continuas [13].

Valor esperado de x. El valor esperado, o media, de una variable aleatoria continua x está definida como

$$E\left[x\right] = \int_{-\infty}^{\infty} x f(x) dx \tag{2.15}$$

y para una variable aleatoria discreta es

$$E\left[x\right] = \sum_{k} x_k p_x(x_k) \tag{2.16}$$

El valor esperado de una variable aleatoria puede ser considerado como la noción intuitiva del promedio de x.

Varianza de x. La varianza es útil cuando se desea conocer qué tanto se extiende una variable aleatoria alrededor de su media.

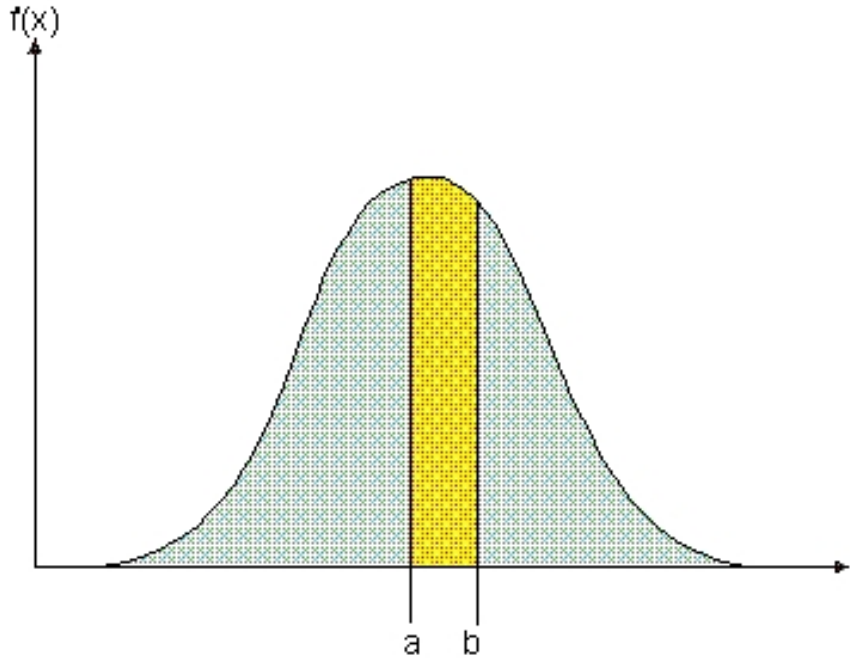

Fig. 2.6: Representación gráfica de la PDF

Sea la desviación de x alrededor de su media $D = x - E[x]$, por lo tanto D puede tomar valores positivos y negativos. Puesto que sólo interesa la magnitud de la variación es conveniente trabajar con $D^2 > 0$. La varianza de la variable aleatoria x está definida como la media cuadrada de la variación $E = D^2$

$$VAR\left[x\right] = E\left[x - E(x)^2\right] \tag{2.17}$$

Tomando la raíz cuadrada de la varianza se obtiene una cantidad en las mismas unidades de x, la que es llamada desviación estándar de la variable aleatoria x

$$STD\left[x\right] = \sqrt{VAR\left[x\right]} \tag{2.18}$$

$STD[X]$ se utiliza como medida del ancho de una distribución. La expresión dada por la Ec(2.17) se puede simplificar a

$$VAR\left[x\right] = E\left[x^2\right] - E\left[x\right]^2 \tag{2.19}$$

El histograma es una herramienta importante en el procesamiento de imágenes. El histograma de una imagen consiste en un diagrama de barras de la propia

imagen, utilizándose como abscisas algún parámetro representativo (una componente RGB, o HSI, o el nivel de gris), y como ordenadas el número de pixeles de la imagen para cada nivel dado. La información que el histograma proporciona sobre la imagen se reduce a la distribución en frecuencia de los diferentes niveles de la componente. En este sentido se pierde toda la información espacial de la imagen.

2.5. Segmentación

La segmentación es el proceso de descomponer una imagen en sus componentes u objetos más básicos. En general la segmentación autónoma es una de las tareas más difíciles en el procesamiento de imágenes. Una segmentación que se considere de buena calidad es aquella que conlleva a una solución exitosa del problema que requiere la extracción de sus partes individualmente. Por el otro lado, una segmentación errática casi siempre garantiza malos resultados o fallos del proceso.

La segmentación divide una imagen en sus regiones u objetos constituyentes, como se mencionó anteriormente. El nivel al cual cada división es llevada depende del problema a resolver, es decir, el proceso de segmentación debería detenerse tan pronto los objetos de interés hayan sido aislados.

Los algoritmos de segmentación generalmente se basan en una o dos propiedades básicas de la componente de intensidad: discontinuidades y similitudes

- *Discontinuidades*, se tiene que una imagen se parte de acuerdo a cambios abruptos en intensidad, como por ejemplo los ejes.

- *Similitudes*, esta categoría está basada en partir a la imagen en regiones que son similares de acuerdo a algún criterio, como por ejemplo, regiones en crecimiento (blobs), umbralización, y fusiones, entre otros. El presente trabajo se enfoca sobre ésta, específicamente sobre la regiones en crecimiento.

Aunque existen diversas aproximaciones para realizar la segmentación de una imagen, en este trabajo sólo se mencionarán tres: k-means, regiones en crecimiento (blobs), y segmentación probabilística.

El primero de ellos, k-means, fue utilizado durante el inicio del proyecto debido al trabajo anterior realizado en [10] con el objeto de reconocer gestos en las personas. Este algoritmo funciona cuando se conocen de antemano el número de clases en las que se van a reconocer en la imagen; cuando ese dato se desconoce, el algoritmo no funciona adecuadamente, además de que si las

semillas no se escogen con cuidado, puede dar lugar a resultados inesperados y erróneos.

Cuando el algoritmo fue implementado en un ambiente dinámico como lo es la Robocup su desempeño no fue el que se esperaba, ya que las regiones (centroides) se perdían en los límites de la imagen, o en movimientos bruscos de los objetos, y no era sencillo retomarlos, por lo que se decidió usar algún algoritmo alterno. Sin embargo se menciona en el presente trabajo debido a que fue implementado y probado (aunque no documentado) y se ha creído conveniente incluirlo. Se utilizó, con adaptaciones al presente trabajo, la técnica utilizada en [2], así como una introspectiva a los métodos probabilísticos, como el clasificador de Bayes.

2.5.1. K-means

Este algoritmo se utiliza cuando se desea dividir en regiones una serie de datos, y ha sido ampliamente aplicado en diversas áreas como reconocimiento de patrones, análisis microbiológico, procesamiento de imágenes satelitales, etc.

El algoritmo *k-means* es un algoritmo de segmentación el cual particiona N entradas (o puntos) $x_1, x_2, \ldots, x_n$ en k regiones asignando una entrada n_t a la jiésima región si se cumple la función

$$I(j|x_t) = 1 \tag{2.20}$$

para $j = argmin_{1 \leq r \leq k} \|x_t - m_r\|^2$, y 0 para cualquier otro caso. Aquí $m_1, m_2, \ldots, m_k$ son llamados semillas que pueden ser determinados en diversas formas, p. ej.

1. Se preasignan k regiones y se inicializan las semillas $\{m_j\}_{j=1}^{k}$
2. Dada una entrada x_t se calcula $I(j|x_t) = 1$
3. Se actualizan las semillas ganadoras m_w, p. ej. $I(w|x_t) = 1$ con

$$m_w^{new} = m_w^{old} + \eta(x_t - m_w^{old}) \tag{2.21}$$

donde η es una tasa pequeña positiva de aprendizaje

Los pasos 2 y 3 se repiten para cada entrada hasta que todas las semillas converjan. En la Fig(2.7) se tiene un conjunto de datos; en la Fig(2.8) se muestra la primera iteración para $k = 5$ y cómo los centroides se agrupan; finalmente en la Fig(2.9) se muestra el último paso donde ya están localizados los centroides.

Aunque este algoritmo es simple y ampliamente utilizado tiene las siguientes desventajas

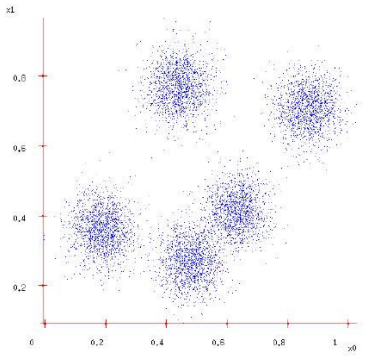

Fig. 2.7: Datos originales

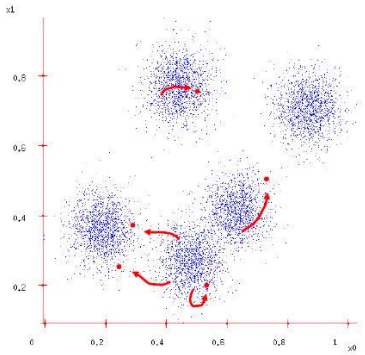

Fig. 2.8: Primera iteración

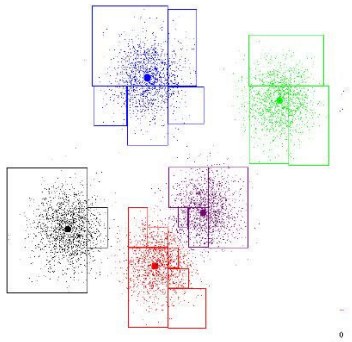

Fig. 2.9: Último paso

- Implica que las regiones tengan forma redonda ya que este se computa basado en la distancia euclidiana.
- Tiene puntos muertos, es decir, si algunas semillas están inicializadas lejos de los datos en comparación con otras semillas, entonces inmediatamente se convierten en puntos muertos sin oportunidad de aprendizaje
- Se necesita predeterminar el número de regiones. Si este no se sabe entonces se corre el riesgo de obtener resultados erróneos.

2.5.2. Agrupación y regiones en crecimiento

Los algoritmos de agrupación convierten los pixeles detectados en una lista de estructuras, por color, que describen cada grupo de pixeles cuatro-conectados. El algoritmo consta de dos pasadas; en la primera convierte la imagen en una lista de elementos RLE (Run Length Encoded). Después con estos elementos se construyen regiones en crecimiento que describen grandes porciones de pixeles interconectados en la imagen, y que a la larga pueden formar objetos. La ventaja de usar RLE's es que el algoritmo se vuelve más simple al trabajar cn grupos de pixeles en lugar de pixeles individuales.

2.5.3. Run Length Encoding (RLE)

El primer paso de este algoritmo es convertir un vector de pixeles en una lista enlazada de elementos RLE's. Este proceso crea una lista de corridas en memoria que encapsulan cada bloque horizontal de pixeles. Un índice lleva la cuenta de las corridas a partir de la esquina superior izquierda.

El sistema RLE (Run Length Encoding) es un algoritmo de compresión sin pérdidas en la cual corridas de datos, es decir, secuencias en las cuales aparecen datos con el mismo valor en muchos elementos consecutivos, son almacenados como un dato simple y su cuenta (en lugar de la secuencia original). Es decir, si un dato d ocurre n veces consecutivas en el flujo de entrada, se reemplazan las n ocurrencias simplemente con el par nd [15]. A n ocurrencias consecutivas de un dato se le llama run-length de n, y a la compresión se le llama encoding.

A pesar de que este algoritmo se originó principlamente para archivos de texto, este es muy utilizado en objetos que contienen muchas de tales corridas, p.ej. imágenes simples como iconos y dibujos con líneas. Como ejemplo se tiene la siguiente corrida en una fila de una imagen que ya ha sido discretizada a 3 regiones, R = rojo, B = azul, G = verde

`RRRRGGGRBBBBBBRRRGG...`

esta quedaría codificada como

`R4G3RB6R3G2...`

donde puede apreciarse que la secuencia original tiene 19 datos mientras que la que ha sido comprimida sólo contiene 11. Sin embargo no es ésta característica del algoritmo RLE la que nos interesa aquí, ya que no se desea comprimir la imagen sino agruparla. El resultado del algoritmo provee una forma simple de almacenar los pixeles adyacentes para la siguiente etapa.

2.5.4. Regiones en crecimiento (Blobs)

Una región es una colección de pixeles conectados, es decir, donde los pixeles se juntan o se tocan [8], y entre dos pixeles cualesquiera conectados de la colección, existe una ruta totalmente definida dentro de ésta, donde una ruta de conexión es una ruta que siempre se mueve entre pixeles vecinos. Por lo tanto, en una región se puede trazar una trayectoria entre cualesquiera dos pixeles sin abandonar nunca la región.

Existen dos reglas de conectividad: four-connectivity y eight-connectivity (se ha mantenido el nombre anglosajón de estas técnicas debido a que se quiere mantener la idea original; así mismo, son estos nombres como se pueden encontrar en la literatura). Para la primera se considera que sólo están conectados los pixeles laterales adyacentes (izquierda, derecha, arriba, abajo), y se dice que los objetos están four-connected. Para el segundo caso, además de los pixeles laterales se toman en cuenta también los pixeles de las diagonales, y se dice que los objetos están eight-connected. En la Fig(3.15) se observa la forma en que se forman los blobs a partir de los RLE's obtenidos anteriormente.

2.5.5. Reconocimiento probabilístico de patrones

Si se desea que un sistema distinga entre objetos de diferentes tipos, primero se debe determinar qué características del objeto deberían ser tomadas en cuenta que produzcan parámetros descriptivos. Las caracterísiticas particulares que han de ser medidas se conocen como características del objeto, y los parámetros resultantes son son los vectores de características. Es importante una buena selección de dichas características puesto que serán las utilizadas para identificar a los objetos.

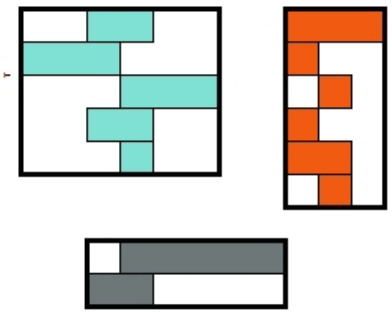

Fig. 2.10: Formación de blobs a partir de RLE's

Existen algunas medios analíticos que guían para la selección de las características, aunque la intuición o la naturaleza intrínseca del problema son las más utilizadas. Las buenas características tienen las siguientes propiedades [8]

- **Discriminación** Los valores que tomen las características deben ser lo suficientemente diferentes para objetos que pertenezcan a diferentes clases
- **Fiabilidad** Los valores que tomen las características deben tomar valores similares para objetos de la misma clase
- **Independencia** Las diferentes características utilizadas deben estar descorrelacionadas entre ellas
- **Longitud** Hay que mantener bajo el número de características ya que la complejidad se incrementa exponencialmente. Aunque los errores pueden reducirse utilizando más características, existe un límite, después del cual el rendimiento se deteriora [9]

El reconocimiento probabilístico de patrones asume que la imagen contiene uno o más objetos y que cada uno de ellos pertence a una de varios tipos predeterminados, o clases. Dada una imagen conteniendo varios objetos, el proceso de reconocimiento de patrones consiste principlamente de tres fases.

En la primera, segmentación, cada objeto es identificado y aislado del resto de la escena.

En la segunda fase, extracción de características, el objeto se mide. Una medida es el valor de alguna propiedad cuantificable del objeto. Una cualidad es una función de una o más medidas, tal que cuantifica alguna característica

del objeto. Esta fase produce una colección de cualidades, que tomadas en conjunto, dan lugar al vector de cualidades. Esto reduce drásticamente la cantidad de información que se utilizará posteriormente para la clasificación del objeto.

En la tercera fase, clasificación, la salida está dada por una regla de decisión que dicta a qué clase pertenece el objeto. Cada objeto se asigna a uno de varios grupos preestablecidos (clases) que representan todos los posibles tipos de objetos que pueden ser hallados en la imagen.

2.5.6. Regla de Bayes

La formas de atacar un problema de reconocimiento están basadas en el uso de funciones de decisión (o funciones discriminantes). Sea $x = (x_1, x_2, \ldots, x_n)^t$ un vector de patrones n-dimensional. Para W patrones de clases $w_1, w_2, \ldots, w_W$ el problema principal es encontrar W funciones de decisión $d_1(x), d_2(x), \ldots, d_W(x)$ con la propiedad de que, si un patrón x pertenece a la clase w_i, entonces

$$d_i(x) > d_j(x) \tag{2.22}$$

con $j = 1, 2, \ldots, W; j \neq i$. Es decir, se dice que un patrón desconocido pertenece a la $i$ésima clase si, dada la substitución de x en todas las funciones de decisión, $d_i(x)$ se toma el que tenga el valor numérico más grande.

La frontera que separa a la clase w_i de la clase w_j está dada por los valores de x para los cuales $d_i(x) = d_j(x)$ o, equivalentemente

$$d_i(x) - d_j(x) = 0 \tag{2.23}$$

Supóngase dos clases y una característica, Fig(2.11). Esto es los objetos pertenecerán ya sea a la clase 1 w_1 o a la clase 2 w_2. Para cada objeto se medira una característica, x.

Podría ser que la PDF de la medida x se conozca para una o ambas clases. Si se desconoce alguna de las PDF's estas deberán ser estimadas a partir de múltiples mediciones sobre la característica x, dibujando el histograma de la característica y luego se calcula la media y la varianza. Después de normalizar a la área unitaria, este histograma puede ser considerado como un estimado de la PDF correspondiente.

La probabilidad de que un objeto x provenga de la clase w_i se denota como $p(w_i/x)$. Si el clasificador decide que x viene de la clase w_j cuando realmente provino de w_i, entonces se incurre en un error, L_{ij}. Dado que un patrón x puede pertenecer a cualquiera de las W clases en consideración, el promedio

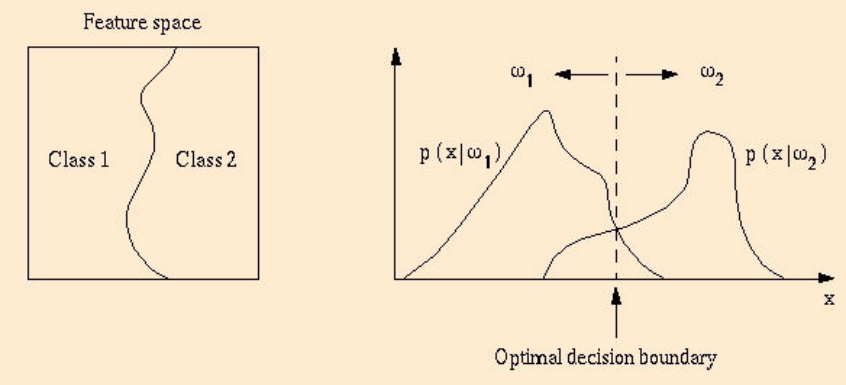

Fig. 2.11: Dos clases y una característica

de las pérdidas incurridas en asignar x a w_j es

$$r_j(x) = \sum_{k=1}^{W} L_{kj} p(w_k|\mathbf{x}) \tag{2.24}$$

Esta ecuación es conocida como el *riesgo condicional promedio*. De teoría de probabilidad se tiene que

$$p(A|B) = \frac{p(A)p(B|A)}{p(B)} \tag{2.25}$$

donde A y B son dos eventos cualesquiera dentro de un mismo espacio de probabilidad. Utilizando esta expresión, la Ec(2.24) se puede escribir como

$$r_j(\mathbf{x}) = \frac{1}{p(\mathbf{x})} \sum_{k=1}^{W} L_{kj} p(\mathbf{x}|w_k) P(w_k) \tag{2.26}$$

donde $p(\mathbf{x}|w_k)$ es la función de densidad de probabilidad de los patrones de la clase w_k y $P(w_k)$ es la probabilidad de ocurrencia de la clase w_k. Dado que $1/p(\mathbf{x})$ es común a todos los términos en $r_j(\mathbf{x}), j = 1, 2, \ldots, W$ éste se puede eliminar de la Ec(2.26) sin alterar el orden relativo de esta función del valor más pequeño añ más grande. Por lo tanto, la expresión para la pérdida promedio queda como

$$r_j(x) = \sum_{k=1}^{W} L_{kj} p(x|w_k) P(w_k) \tag{2.27}$$

$p(x|w_k)$ representa la densidad de probabilidad de que el evento x pertenezca a la clase w_k, mientras que $P(w_k)$ es la probabilidad de la clase w_k (la p es densidad de probabilidad, y P es la probabilidad del evento).

El clasificador tiene W posibles clases a escoger dado un patrón de entrada cualquiera. Si se calcula $r_1(x), r_2(x), \ldots, r_W(x)$ para cada patrón x y se asigna el patrón a la clase con la menor pérdida, la pérdida total promedio con respecto a todas las decisiones será mínimo. El clasificador que minimiza esta pérdida es el llamado *Clasificador de Bayes (CB)*. Así el CB asigna a un patrón desconocido x a la clase w_i si $r_i(x) < r_j(\mathrm{x})$ para $j = 1, 2, \ldots, W; j \neq i$. Es decir, x es asignado a la clase w_i si

$$\sum_{k=1}^{W} L_{ki} p(\mathbf{x}|w_k) P(w_k) < \sum_{q=1}^{W} L_{qj} p(\mathbf{x}|w_q) P(w_q) \tag{2.28}$$

para toda $j; j \neq i$. A la "pérdida"para una decisión correcta se le da el valor de cero, mientras que la pérdida para una decisión incorrecta toma un valor diferente de cero, p. ej. uno. Bajo estas consideraciones la función de pérdida se transforma en

$$L_{ij} = 1 - \delta_{ij} \tag{2.29}$$

donde $\delta_{ij} = 1$ si $i = j$ y $\delta_{ij} = 0$ si $i \neq j$. La Ec(2.29) indica una pérdida de uno para decisiones incorrectas, y una pérdida de cero para decisiones correctas. Substituyendo la Ec(2.29) en la Ec(2.27) se tiene

$$\begin{aligned} \mathrm{r_j}(x) &= \sum_{k=1}^{W} (1 - \delta_{kj}) p(x|w_k) P(w_k) \\ &= p(x) - p(x|w_j) p(w_j) \end{aligned} \tag{2.30}$$

Entonces el CB asigna a un patrón x a las clase w_i, si, para toda $j \neq i$

$$p(x) - p(x|w_i) P(w_i) < p(x) - p(x|w_j) P(w_j) \tag{2.31}$$

o equivalentemente

$$p(x|w_i) P(w_i) > p(x|w_j) P(w_j) \text{para toda} j = 1, 2, \ldots, W; j \neq i \tag{2.32}$$

2.6. Estructuras de datos: Listas enlazadas

Una estructura de datos es cualquier colección o grupo de datos organizados de tal forma que tengan asociados un conjunto de operaciones para poder manipularlos.

Una lista enlazada es una estructura de datos que tiene una organización lineal y se caracteriza porque cada uno de sus elementos tiene que indicar dónde se encuentra el siguiente elemento de la lista. La estructura de la lista es lineal, esto es, existe un elemento llamado "'el primero"' y otro llamado "'el último"'; cada elemento tiene un único sucesor y antecesor; cada uno de los elementos tiene asignada una única posición en la lista. En la Fig(2.12) se tiene una lista enlazada de tres elementos.

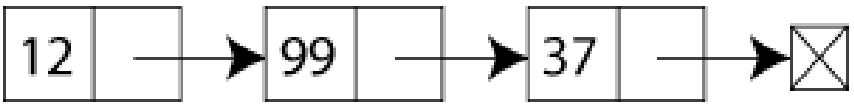

Fig. 2.12: Lista enlazada

La lista enlazada puede representarse en memoria estática utilizando un arreglo de nodos, o más comúnmente, con memoria dinámica utilizando nodos encadenados a través de apuntadores. Además debe de existir un apuntador principal, externo a la estructura, que almacene la dirección del primer elemento de la lista, ya que a partir de él, se encadenará el resto de los elementos.

Las listas enlazadas se utilizan para dos propósitos principales. El primero es crear arreglos de tamaño desconocido en memoria. Si se conoce previamente la cantidad de almacenamiento, se puede utilizar un simple array; pero si no se conoce el tamaño real de una lista, entonces hay que utilizar una lista enlazada. La segunda utilización es en los archivos de almacenamiento en disco para bases de datos. Las listas enlazadas permiten insertar y borrar elementos rápida y fácilmente sin volver a reoganizar el archivo completo de disco.

Las listas enlazadas se utilizan en las etapas de Segmentación y Agrupación. Para la primera se tiene que cada nueva corrida RLE se almacena en una estructura que se liga a la corrida anterior, y este proceso se repite mientras se lee la imagen completa. En el segundo caso cuando se agrupan los RLE's y forman un BLOB esta nueva información es ligada al BLOB formado anteriormente, y este proceso se repite hasta que se alcance el final de la lista de corridas RLE.

2.7. Comunicaciones: Sockets de internet

TCP/IP es el protocolo común utilizado por todos los ordenadores conectados a Internet, de manera que puedan comunicarse entre sí. Hay que tener en cuenta que en Internet se encuentran conectados ordenadores de clases muy diferentes y con hardware y software incompatibles en muchos casos, además

de todos los medios y formas posibles de conexión. Aquí se encuentra una de las grandes ventajas del TCP/IP, pues este protocolo se encargará de que la comunicación entre todos sea posible. TCP/IP es compatible con cualquier sistema operativo y con cualquier tipo de hardware.

Para transmitir información a través de TCP/IP, ésta debe ser dividida en unidades de menor tamaño. Esto proporciona grandes ventajas en el manejo de los datos que se transfieren y, por otro lado, esto es algo común en cualquier protocolo de comunicaciones. En TCP/IP cada una de estas unidades de información recibe el nombre de "datagrama", y son conjuntos de datos que se envían como mensajes independientes. TCP (Transmission Control Protocol).

Cuando la información se divide en datagramas para ser enviados, el orden en que éstos lleguen a su destino no tiene que ser el correcto. Cada uno de ellos puede llegar en cualquier momento y con cualquier orden, e incluso puede que algunos no lleguen a su destino o lleguen con información errónea. Para evitar todos estos problemas el TCP numera los datagramas antes de ser enviados, de manera que sea posible volver a unirlos en el orden adecuado. Esto permite también solicitar de nuevo el envío de los datagramas individuales que no hayan llegado o que contengan errores, sin que sea necesario volver a enviar el mensaje completo.

Cada proceso que se desea comunicar con otro se identifica en la pila de protocolos TCP/IP con uno o más puertos. Un puerto es un número de 16 bits, empleado por un protocolo host a host para identificar a que protocolo del nivel superior o programa de aplicación se deben entregar los mensajes recibidos.

Los puertos bien-conocidos pertenecen a servidores estándar, y se hallan en el rango de 1 a 1023, por ejemplo Telnet usa el puerto 23 . Estos puertos suelen tener números impares, debido a que los primeros sistemas que usaron el concepto de puerto requerían para las operaciones en duplex una pareja par/impar de puertos. La mayoría de los servidores requieren un único puerto.

Los clientes no necesitan puertos bien-conocidos porque inician la comunicación con los servidores y los datagramas UDP enviados al servidor contienen su número de puerto. El host en funcionamiento proporciona un puerto a cada proceso cliente mientras este lo necesite. Los números de puertos efímeros tienen valores mayores de 1023, por lo general en el rango de 1024 a 5000. Un cliente puede usar cualquier número en ese rango, siempre que la combinación *protocolo de transporte, dirección IP, número de puerto* sea unívoca.

La Fig(2.13) muestra la forma de implementar sockets *servidor* y *cliente* en lenguaje C. Ambas partes deben coincidir en el tipo de comunicaciones, TCP o UDP. Así mismo, el *cliente* debe conocer el puerto por el que el *servidor* está escuchando. Antes de que un *cliente* pueda conectarse, el *servidor* ya debe

estar listo y a la escucha de peticiones de conexión. El software desarrollado se considera como *cliente*, por lo que la máquina responsable de procesar la información de visión se configura como *servidor*.

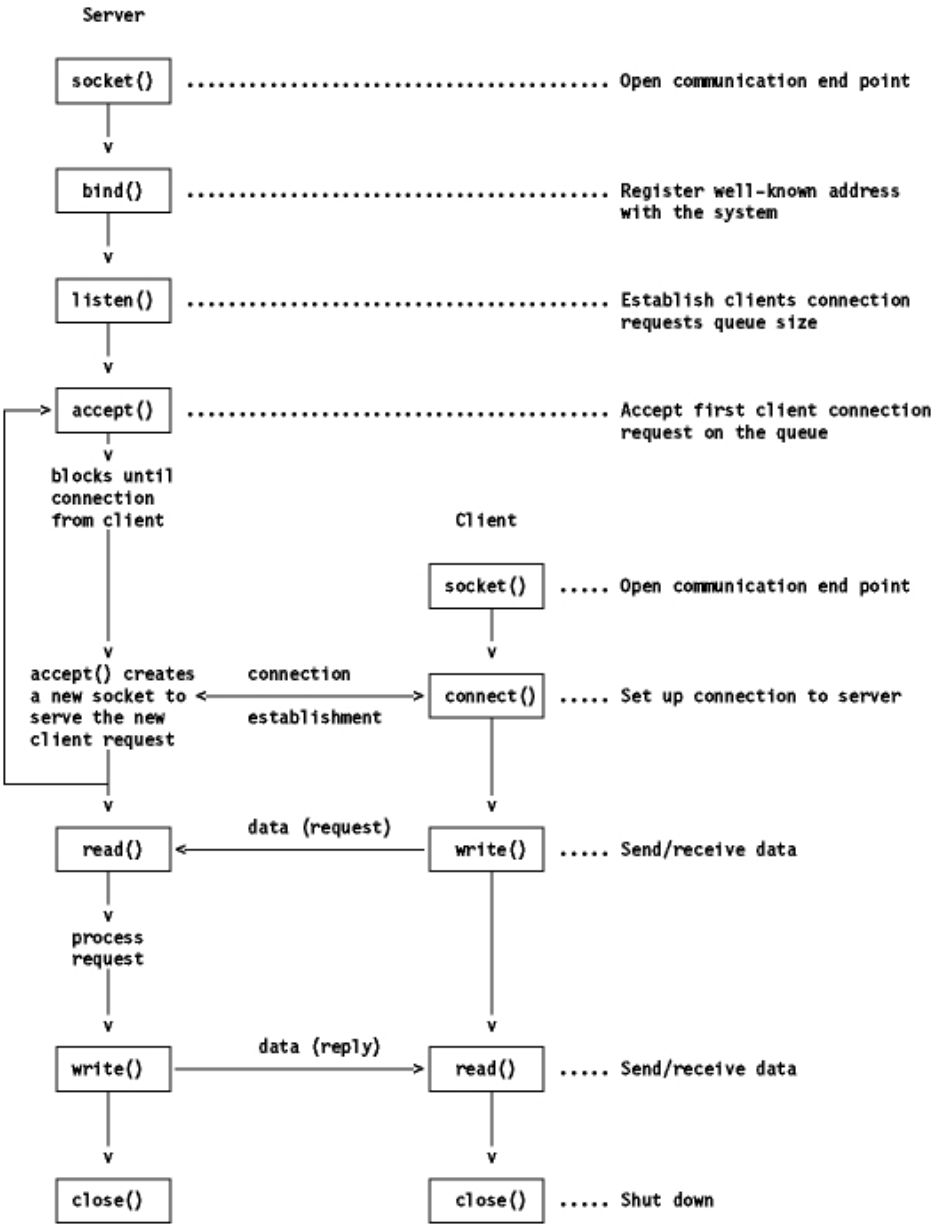

Fig. 2.13: Implementación de sockets

3. DESARROLLO

3.1. Software

En el capítulo anterior se habló de manera extensa acerca de todo el proceso que la visión robótica implica, desde la captura de la imagen hasta el reconocimiento de los robots dentro de la escena. Para facilitar la forma en que el programdor y el usuario interactúan con el sistema se desarrolló una GUI (Graphical User Interfaz, Interfaz Gráfica de Usuario), la cual contiene todos los parámetros necesarios para la configuración de cada etapa. Aunque es posible encadenar todos los subprocesos en comandos tipo batch, o por lote (.bat) se prefirió utilizar las API que los sistemas operativos proveen a los programadores, ya que de esta forma un usuario no experimentado o ajeno al proyecto puede utilizar la aplicación de manera relativamente fácil, ya que la interfaz es simple e intuitiva.

Una idea central alrededor del desarrollo de esta aplicación es la separación del modelo (los algoritmos de procesamiento de imágenes) de su vista (lo que el usario ve) y de sus controles (las entradas que el usuario da por teclado y ratón), es decir, que estos algoritmos no estén casados con el sistema operativo en el que fueron desarrollados, sino que se puedan portar a otras plataformas. Así, si alguien requiere de los algoritmos de preprocesamiento únicamente tomaría el código de estos y los utilizaría en su aplicación, sin la necesidad de incrustar la suya en el software desarrollado aquí.

Las herramientas utilizadas fueron Visual Studio como compilador de C, apoyándose con las MFC (Microsoft Fundation Clases), las cuales son una capa entre las API de Windows y el programador, facilitando la programación de la GUI. Se decidió utilizar este compilador debido a la facilidad para implementar las interfaces de usuario, así como que se iba a utilizar DirectX como interfaz entre el hardware de las cámaras y la aplicación. Por lo mismo la aplicación se desarrolló bajo el sistema operativo Windows XP. Una práctica común entre la comunidad es el uso del sistema operativo Unix y Linux para este tipo de aplicaciones, sin embargo, cuando no se conoce éste sistema operativo las cosas no resultan ser tan fáciles como en Windows, ya que habría que aprender a utilizar herramientas específicas de este entorno como las librerías de gráficos

y de controles (para la construcción de la GUI), así como librerías de captura, que no siempre son fáciles de encontrar y configurar, por lo que se dió dar prioridad al resultado y no a la forma. La Fig(3.1) muestra la aplicación, donde pueden apreciarse todas las etapas del proceso de visión.

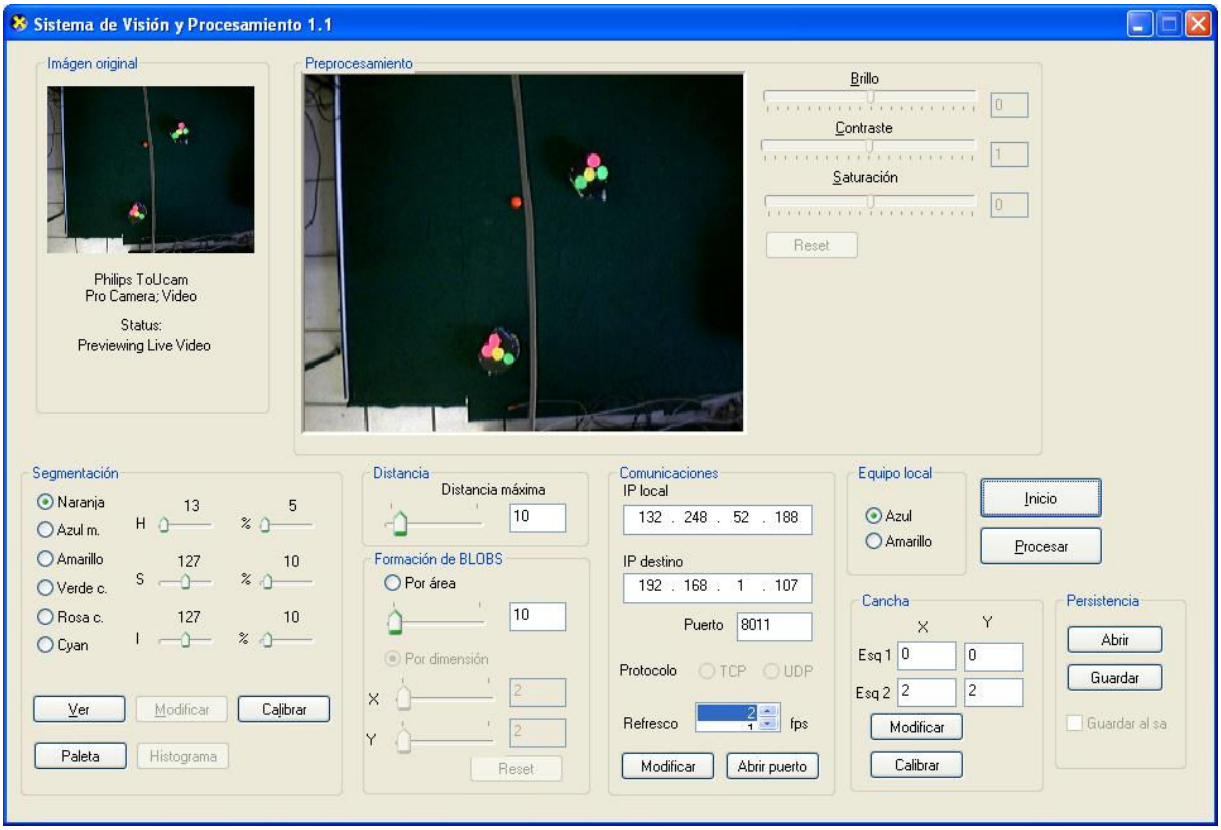

Fig. 3.1: Aplicación para el reconocimiento de los robots

3.2. Captura

3.2.1. DirectX

Microsoft DirectX es una colección de interfaces para programar aplicaciones multimedia basadas en el sistema operativo Windows. Esta interfaz provee una plataforma de desarrollo estándar para computadoras tipo PC permitiéndole a los programadores acceder a características especializadas del hardware sin tener que escribir código específico para ese hardware. Esta tecnología fue introducida en 1995 y se ha convertido en un estándar para el

desarrollo de aplicaciones multimedia sobre la plataforma Windows.

En el núcleo de DirectX se encuentran las interfaces para programacion de las aplicaciones (API's). Estas API's actúan como puente entre el hardware y el software para que se comuniquen entre ellas. Así mismo, estas ofrecen acceso a las aplicaciones multimedia y características avanzadas de alto rendimiento como por ejemplo: los chips de aceleración para gráficas en 3D y tarjetas de sonido; soportes de entrada como palancas de mando, teclados, ratones, etc; control de la mezcla y salida del sonido.

El kit de desarrollo de DirectX es distribuído gratuitamente por Microsoft . Las librerías de DirectX eran originalmente distribuídas por los desarrolladores de juegos con sus paquetes pero más tarde fueron incluídas en Windows. Su última versión es la 9.0. DirectX incluye las siguientes APIs:

- DirectDraw: para dibujado de imágenes en dos dimensiones (planas).
- Direct3D (D3D): para representación de imágenes en tres dimensiones.
- DirectInput: utilizado para procesar datos del teclado ratón joystick y otros controles para juegos.
- DirectPlay: para comunicaciones en red.
- DirectSound: para la reproducción y grabación de sonidos de ondas.
- DirectMusic: para la reproducción de pistas musicales compuestas con DirectMusic Producer.
- DirectShow: para reproducir audio y video con transparencia de red.
- DirectSetup: para la instalación de componentes DirectX.

DirectShow divide el procesamiento de los procesos multimedia, p. ej. reproducir video, en pasos conocidos como filtros. Los filtros tienen terminales de entrada y salida para conectarse entre ellos. El diseño genérico de la interfaz permite que los filtros sean conectados en diferentes formas para obtener diversos resultados. DirectShow es ampliamente utilizado para reproducción de video (los filtros realizan las tareas de parseo, demultiplexado de audio-video, descompresión y rendering), así como grabación y edición de audio.

DirectShow conecta en cadena los filtros de forma que la salida de cada filtro es la entrada del siguiente. El *Filter Graph Manager* es el encargado de gestionar la construcción del árbol de filtros, él insertará los filtros directa o indirectamente en el grafo (un grafo es una agrupación de filtros). El *Filter*

Graph Manager es un objeto COM, accesible mediante la interfaz IGraph-Builder. Este reside en una DLL almacenada en la computadora. Es decir, el Graph Manager no pertenece a ninguna aplicación en particular, sino que se encuentra en la PC y es accesible desde cualquier aplicación.

La aplicación utilizará la funcionalidad de DirectShow mediante mensajes enviados a ese objeto COM. Estos mensajes serán llamadas a métodos Direct-Show y éste responderá a través de eventos, como lo muestra la Fig(3.2)

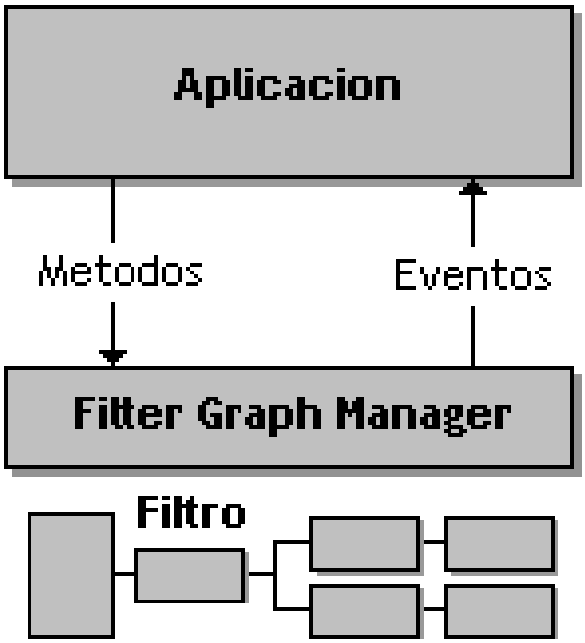

Fig. 3.2: Arquitectura de DirectShow

Dado que por sí misma esta interfaz es bastante compleja y su estudio está fuera de los alcances del presente trabajo, sólo bastó con interceptar el búfer que contiene a la imagen capturada. En el software realizado se tiene una variable $pBuffer(x)$ que es la que almacena el búfer RGB en forma vectorial, por lo que más adelante habrá que realizar la conversión de vector a matriz. Para pintar en la pantalla existen dos maneras: pintando pixeles utilizando la API de Windows, o escribiendo en el búfer $pBuffer(x)$. Por velocidad el dibujado se hará utilizando dicho búfer, por lo que otra vez se habrá de regresar de forma matricial a forma vectorial.

3.2.2. Cámaras

Se están utilizando *webcams* como sensores de imágenes debido a su bajo costo y su fácil consecución, aunque su calidad a veces deja mucho que desear. La calidad de la imagen entregada depende de la marca, y aún para dispositivos del mismo fabricante, ésta puede variar sensiblemente. Sin embargo, como se mencionó en el capítulo 1, el uso de este tipo de cámaras permitirá que personas e instituciones con bajo presupuesto puedan tener acceso a este tipo de proyectos sin tener que desembolsar grandes cantidades de dinero, como sería el caso de cámaras digitales, ya sea para un proyecto de las dimensiones del que nos ocupa o como mero pasatiempo y/o introducción al procesamiento de imágenes en movimiento y robots móviles. En la Fig(3.3) se muestra el montaje de una de las cámaras.

Fig. 3.3: Montaje de una de las cámaras web utilizadas en el proyecto

La resolución de una webcam típica va desde los 160*120 pixeles, hasta las de 640*480, y algunas alcanzan los 1024*800 en modo interpolado. A mayor resolución mayor tiempo de procesamiento, ya que el búfer crece a una razón no lineal dependiente del producto L*H, donde L es largo de la imagen y H es el ancho, ambos dados en pixeles. Obviamente entre más pixeles se tengan mejor será el detalle capturado, pero se llega a un punto de compromiso donde habrá qué elegir que conviene más al proyecto. Para el presente trabajo se tiene que las cámaras se han puesto a una resolución de 320* 240 pixeles, y

los datos en crudo son enviados a la PC por un puerto USB.

Dado que la altura a la que están las cámaras es de 3.35 m, una sola no puede ver el campo de juego completo, por lo que es necesario utilizar una cámara para cada mitad del campo. Entonces es necesario abrir dos instancias de la aplicación (una para cada cámara) y calibrarlas por separado. La Fig(3.4) muestra la vista de una de las cámaras .

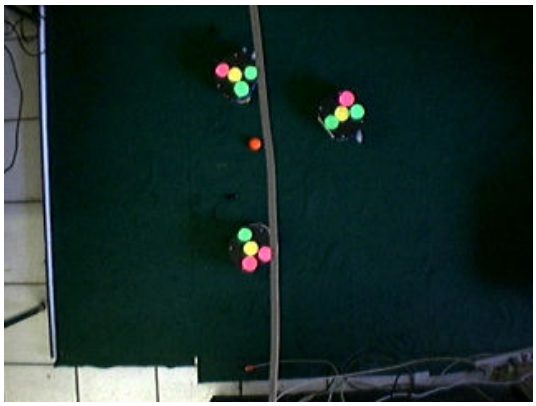

Fig. 3.4: Vista aérea de los robots

3.2.3. Conversión

La interfaz entrega un vector lineal $I_{RGB}(x)$ el cual está compuesto de los elementos R, G, B en forma consecutiva $I_{RGB}(x) = r_0 g_0 b_0, r_1 g_1 b_1, \ldots, r_M, g_M, b_M$ con M dado por $(L-1)(H-1)$, ambas dimensiones en pixeles. Para una cámara en color que entrega $320 * 240$ pixeles, se tiene un total de *230,400* pixeles, donde cada uno de ellos ha de ser procesado más adelante.

Este vector $I_{RGB}(x)$ es difícil de manejar y manipular debido a que las diferentes componentes están entrelazadas, por lo que es necesario y conveniente, dado que es más intuitivo, convertirlo a forma matricial y separarlas en tres, una para cada componente: $I_R(x,y), I_G(x,y), I_B(x,y)$

$$I_B(x,y) = \sum_{y=0}^{YMAX} \sum_{x=0}^{XMAX} I_{RGB}(3x + yYMAX) \tag{3.1}$$

$$I_G(x,y) = \sum_{y=0}^{YMAX} \sum_{x=0}^{XMAX} I_{RGB}(3x + yYMAX + 1) \tag{3.2}$$

$$I_R(x,y) = \sum_{y=0}^{YMAX} \sum_{x=0}^{XMAX} I_{RGB}(3x + yYMAX + 2) \tag{3.3}$$

donde $XMAX$ y $YMAX$ son las coordenadas máximas para ambos ejes. Nótese que se empezó con la componente azul B debido a que ese es el orden entregado por la interfaz. Este es el juego de vectores que se procesará más adelante.

Coordenadas lógicas vs coordenadas reales. El sistema coordenado será el utilizado comúnmente cuando se trabaja con pantallas de computadora, es decir, la esquina superior izquierda será el orígen $(0, 0)$, la abscisa se moverá en sentido positivo hacia la derecha, y la ordenada se moverá en sentido positivo hacia abajo, como lo muestra la Fig(3.5).

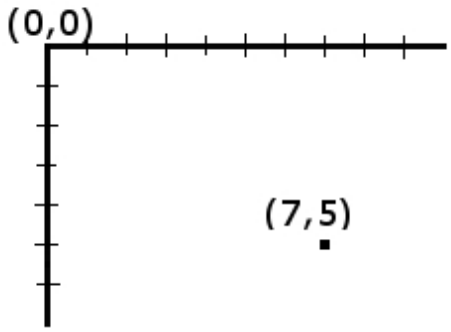

Fig. 3.5: Sistema de coordenadas

3.3. Preprocesamiento

Esta segunda etapa se encarga de mejorar las condiciones de la imagen entregada en bruto de la sección anterior. Es decir, la imagen podría llegar con ruido, obscura, borrosa, etc, por lo que habría que tratarla convenientemente para no acarrear errores tanto en esta etapa como en las subsecuentes.

Es importante apuntar que dada la experiencia lograda a través del desarrollo del proyecto se ha visto que no es necesario agregar ningún tipo de filtrado, debido a dos situaciones: por un lado se tiene que insertar un (o varios)

filtro(s) consumiría tiempo, y por el otro la imagen entregada es lo suficientemente limpia para poder trabajar con ella directamente. Sin embargo, si se requiriese más adelantees es posible intercalar un filtro en este punto.

Otro hecho importante es que en esta etapa se debe realizar la corrección geométrica de las lentes de gran angular. No obstante, ya que las *webcams* no hacen uso de este tipo de lentes, no se ha recurrido a ello. Estas lentes se utilizan mayormente con cámaras analógicas, del tipo Vidicón, y con cámaras digitales. Por lo tanto, si en un futuro se utilizan dichas lentes, la correción geométrica deberá ser puesta en este lugar.

La principal problemática que se tiene es con respecto a la iluminación del ambiente donde se desarrollan las competencias, así como la del laboratorio. Por ello es imperativo controlar los diversos aspectos de la percepción de las imágenes para equilibrar estas diferencias de iluminación y adecuarlas a un momento en particular. Como se vió del capítulo anterior, los parámetros que podrían ser modificados son el brillo, el contraste y la saturación, características ya fueron comentadas , por lo que no es necesario repetirlas, sino que se mostrará la forma en que se implementaron.

3.3.1. Atributos de la imagen

El software cuenta con tres controles con los que se regula la cantidad de brillo, contraste y saturación dada a la imagen, (Fig(3.6)). Cada control va desde 0 hasta 2^8, lo que significa que en algún momento se puede sobrepasar el límite, tanto para arriba como para abajo, de una palabra de 8 bits cuando se le sume a la componente el valor del control, por lo que la rutina se encarga de ajustar los valores para que ello no suceda.

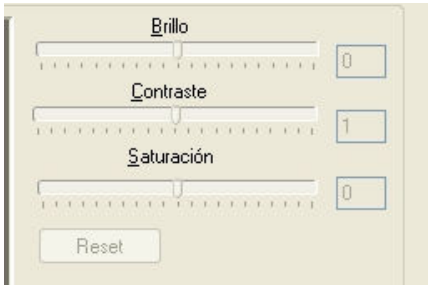

Fig. 3.6: Controles de Brillo, Contraste y Saturación

3.3.2. Conversión de espacios de color

Antes de realizar la segmentación de la imagen es necesario cambiarla de espacio de color; para ello se toma cada pixel RGB y se transforma a su equivalente HSI. La utilización de éste espacio logra una mejora substancial en la separabilidad de clases como se explicará en los siguientes párrafos.

Las ventajas del espacio HSI contra el espacio RGB están dadas por la separación de la componente de color H, lo que permite que se puedan segmentar los colores de forma muy eficiente; también se tiene que, en una competencia real, una situación adversa recurrente es el cambio de luz del campo de juego, por lo que la componente de luminosidad I fácilmente se puede adaptar para contrarrestar dicho efecto. La Fig (3.7) muestra la forma en como se comportan ambos espacios ante la diferencia de luz. La figura de la izquierda muestra un color de ejemplo con sus respectivos valores para HSI y RGB y una cierta cantidad de luz. La figura de la derecha muestra el mismo color pero con menos luz. Nótese la variación de las componentes RGB ante dicha variación de luz, mientras que para las componentes HSI únicamente varió I (luminosidad). Esta propiedad permite que el software desarrollado se adapte con facilidad a los cambios de luz.

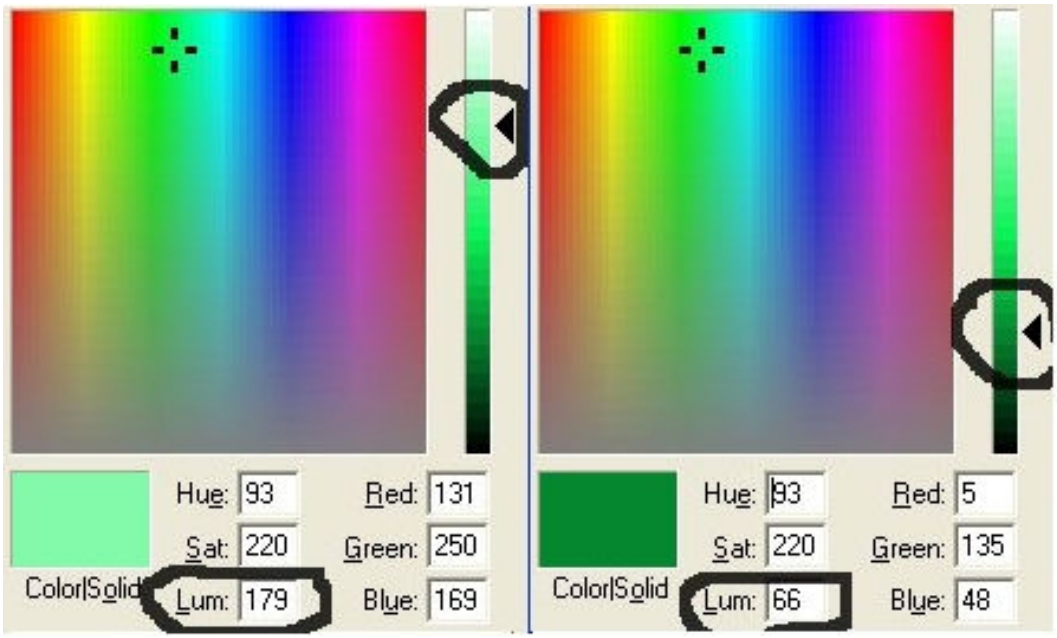

Fig. 3.7: Componentes HSI y RGB ante un cambio de luminosidad

La Fig (3.8) ejemplifica lo que sucede cuando el color va de muy vivo a gris

(este efecto se da cuando los parches están sucios o la cámara está muy alejada); sin embargo, el color sigue siendo el mismo como lo muestran la componente H de ambas partes de la figura. Como sucedió en el ejemplo anterior, la variación para las componentes RGB son muy grandes y poco manejables para calibrar a los colores.

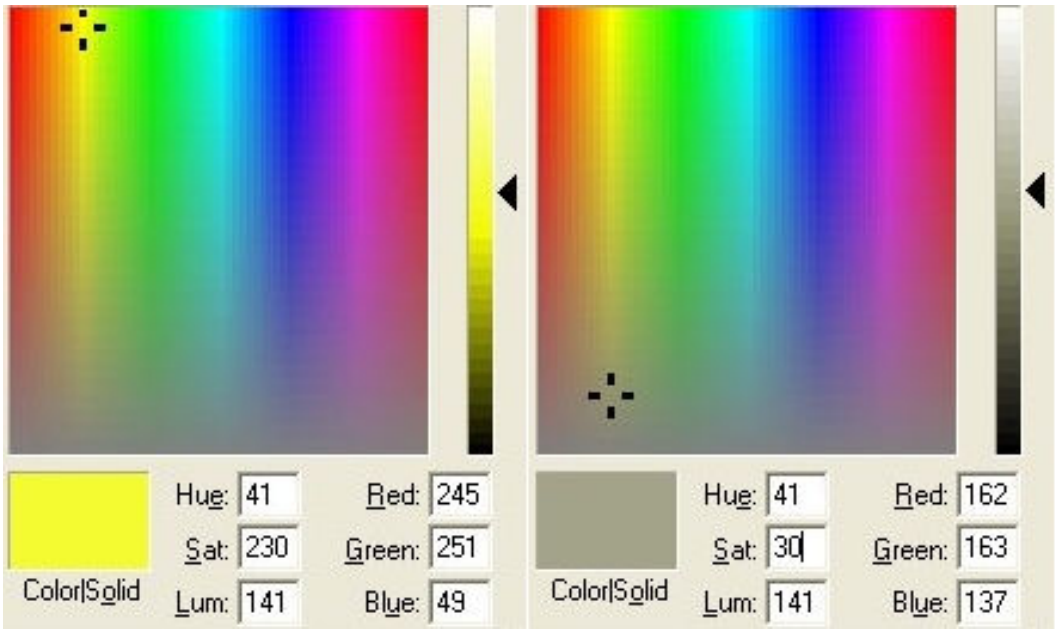

Fig. 3.8: Componentes HSI y RGB ante un cambio en la saturación

Finalmente se tiene la componente más importante, H, ya que de ella depende la correcta segmentación del universo de clases de colores. Continuando con el ejemplo, la Fig (3.9) muestra la variación de las componentes de ambos espacios ante un cambio en el color. La figura de la izquierda muestra un color de ejemplo, y la de la derecha otro color, pero mantentiendo constantes la saturación y la luminosidad. Nótese que sólo varió la componente H del espacio HSI, mientras que para el espacio RGB las tres componentes cambiaron con un márgen muy grande, por lo que éste efecto hace muy poco manejable la separabilidad de colores.

La Fig (3.10) muestra la separación de las diferentes clases de colores basado en la componente H. Se puede apreciar que se diferencían casi perfectamente, aunque en la práctica existen traslapes principalmente entre el rosa, naranja y amarillo. No obstante, el resultado que se obtiene es mucho mejor que con el espacio RGB.

La Fig (3.11) muestra una comparación entre ambos espacios. Los datos

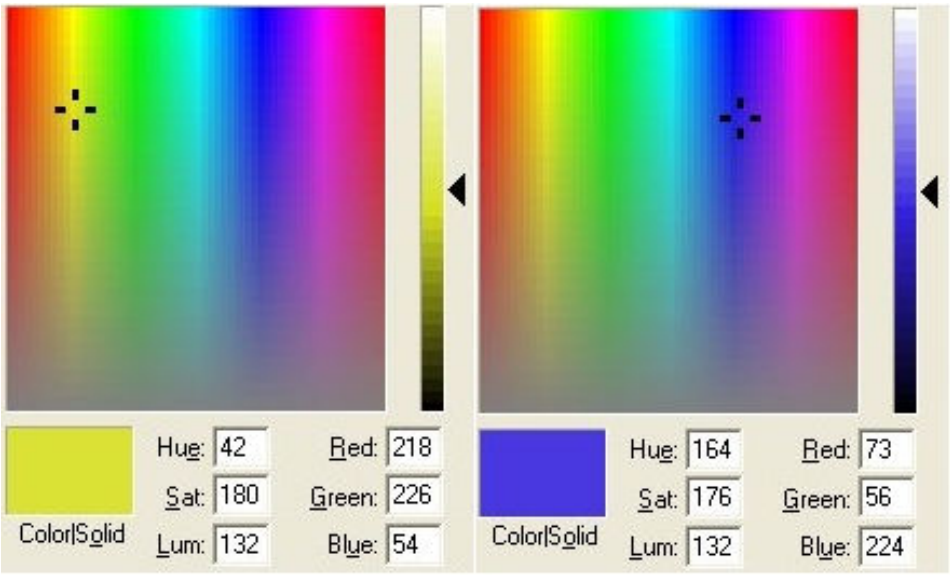

Fig. 3.9: Componentes HSI y RGB ante un cambio en el color o matiz

fueron recogidos de valores reales dados por el submódulo de calibración de la aplicación desarrollada. El principal punto a mencionar es la relativa facilidad para separar las clases utilizando simplemente la componente H, y utilizando S e I (en ese órden) como características auxiliares para la correcta separabilidad de las clases. Para el espacio RGB la segmentación es impráctica debido al esparcimiento de los valores dados para cada componente. Así mismo, si la iluminación cambia habría que recalibrar todas las componentes para todos los colores, lo cual lo hace aún más impráctico.

Dado que la conversión no es lineal debido la función trigonométrica $cos^{-1}(x)$, en lugar de computar este factor para cada pixel, se calcula una tabla de 2000 entradas al inicio de cada corrida de la aplicación, y posteriormente se realizan búsquedas en esta tabla.

Otra simplificación es el uso del factor 765 en la Ec(2.8), con el fin de reducir operaciones, ya que de otra manera hubiera sido necesarias algunas divisiones y normalizaciones. Con esta constante, la cual es la suma de los máximos de los tres valores, (255*3), se logra el mismo resultado, pero evitando dichas operaciones extras.

3.3.3. Autocalibración

La autocalibración se refiere al procedimiento para definir numéricamente a las clases, es decir, es aquí donde se definen los límites dentro del cono HSI

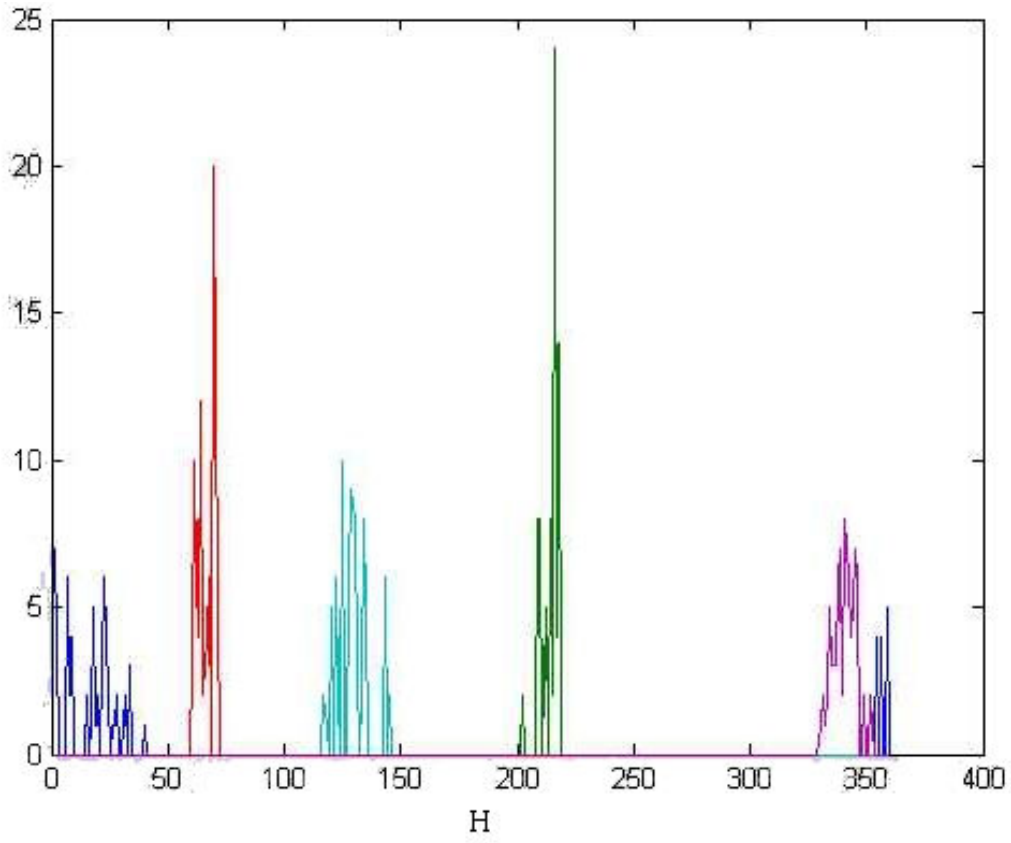

Fig. 3.10: Separabilidad de clases utilizando la componente H

para cada clase de color.

Para ello, la aplicación cuenta con 7 opciones, una para cada color a reconocer, y 6 controles, 3 de los cuales estarán centrados en la región del color (un centro para cada componente HSI), y 3 controles que definirán la apertura de una ventana para cada región.

Dado que es difícil y engorroso calibrar manualmente cada color, este módulo cuenta con un sistema denominado de *autocalibración*, el cual entrena al programa (mediante unas pulsaciones del ratón sobre tres parches del mismo color distribuídos por la cancha). En cada click se toma el histograma de una ventana de 3*3 pixeles, y luego se combinan los 3 histogramas para obtener el valor esperado $E\{x\}_i$ y la desviación estándar σ_i. $E\{C_i\}$, donde C_i es la iésima componente con $i = 1, 2, 3$, se utiliza para centrar la componente en el cono HSI, mientras que σ_i define la apertura de la ventana. La idea principal detrás de esto es que el programa pueda abarcar tanto como sea posible para

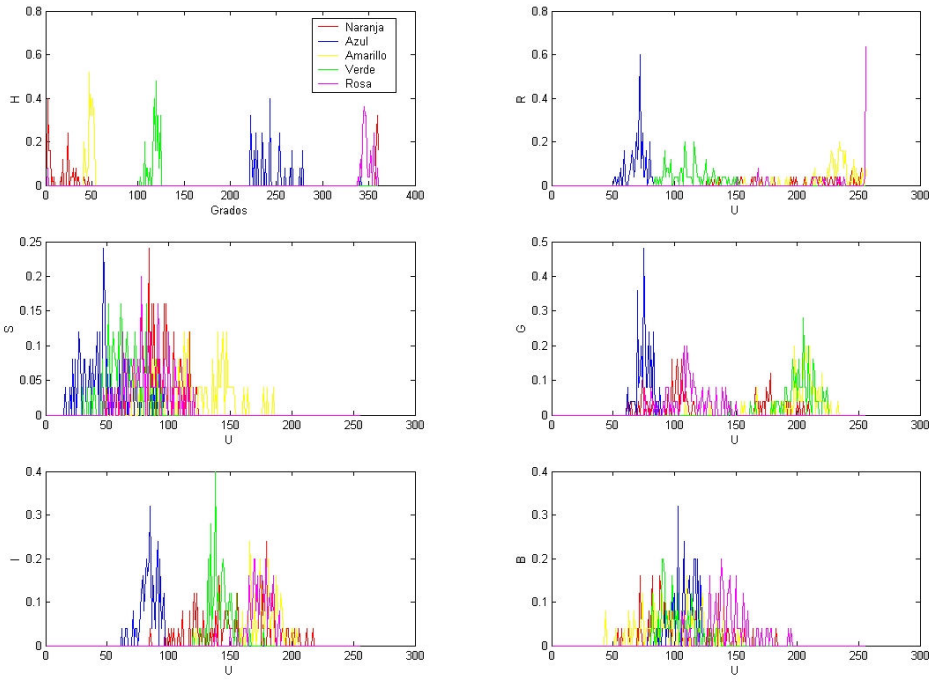

Fig. 3.11: Comparación de los espacios HSI y RGB para la aplicación desarrollada

cada componente, debido a que en una zona de la cancha se obtienen unos resultados, mientras que en otra parte de la misma cancha se tienen resultados ligeramente diferentes, por lo que de dejarlo así cuando el robot entre a un área con características diferentes (mayoritariamente de iluminación) ya no podrá reconocer a los colores. Con el sistema propuesto se pretende obtener los límites superior e inferior que definan al rango de la componente sobre toda la cancha. La Fig(3.12) muestra los controles.

3.4. Segmentación

Una vez que los rangos para cada color han sido definidos, se pasa a la siguiente etapa, la cual consiste en discriminar a cada uno de los pixeles y clasificarlos en una de las 8 clases (7 colores y una no válida) que se deben reconocer.

El proceso de segmentación consiste en mapear a todos y cada uno de los pixeles presentes en la imagen. Este proceso ha de entenderse como el proceso de asignar a cada pixel uno de varios grupos, o clases, bien definidos de colores. Esto es similar a la discretización de una señal analógica. Con 8 bits

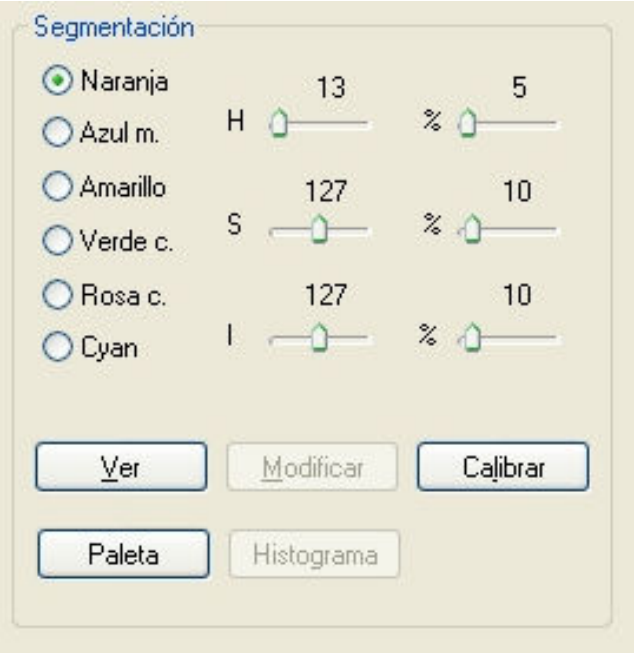

Fig. 3.12: Controles que definen los rangos para la cuantización de los pixeles

se tienen 256 niveles, lo que significa que cualesquier valor que tenga la señal de entrada deberá ser mapeada en uno de estos valores discretos; de manera similar, si se tienen 8 regiones, clases o niveles de color, entonces un pixel deberá caer forzosamente dentro de una de éstas. Sin embargo, este proceso es más complejo que el de discretizar una señal analógica, dado que son valores vectoriales, a diferencia de los valores escalares de una señal analógica simple. Por lo tanto, en realidad han de mapearse 3 señales independientemente y luego relacionarlas entre sí.

El método utilizado es simple y directo, no hace uso de información estadística que sobrecargue al sistema, y consiste en verificar si cada compoente HSI del pixel se encuentra dentro de la ventana definida por la autocalibración (el centroide dado por el valor esperado $E\{C_n\}$, y la apertura de la ventana dada por la desviación estándar σ_n) para cada color (o clase). Si el pixel está dentro de la ventana de la clase n, entonces es clasificado como perteneciente a esa clase, y se crea o agrega a un RLE (ver más adelante). Si el pixel está fuera de la la clase C_n, entonces se pasa a comparar con C_{n+1}, y así sucesivamente hasta completar las 7 clases. En caso de que no haya pertenecido a ninguna clase, el pixel es desechado, y el proceso se repite hasta completar todos pixeles de la imagen.

```
PARA cada pixel en la imagen
  PARA cada clase C; n=0; n++
    SI( LowerLimC0H < HComp <= UpperLimC0H )
      SI( LowerLimC0S < SComp <= UpperLimC0S )
        SI( LowerLimC0I < IComp <= UpperLimC0I )
          ENTONCES el pixel pertenece a la clase Cn
        FIN SI
      FIN SI
    FIN SI
  FIN SI
FIN PARA
```

donde

- **xComp** es la componente a comparar
- **LowerLimC0x** es el límite inferior de la ventana para x=H,S,I
- **UpperLimC0H** es el límite superior de la ventana para x=H,S,I

3.4.1. Agrupamiento de pixeles: RLE's

Cada vez que un pixel es asignado a la i-ésima clase C_i, éste pasa a formar parte en una lista enlazada correspondiente a su clase, por lo que existen 7 listas enlazadas. Los campos de la lista son

- Color
- Posición inicial y final del RLE
- Renglón
- Tamaño
- Identificador
- Enlace al siguiente nodo

El campo ***Color*** es una constante definida para cada uno de los colores; ***Posición inicial y final del RLE*** son la columna x_0 donde inicia el RLE, y x_1, la columna donde finaliza. Recuérdese de la discusión acerca de RLE's del capítulo anterior que el algoritmo va agrupando elementos similares, por lo que si la imagen contiene 7 pixeles consecutivos de, p. ej, color rojo y a continuación 3 de color verde, la corrida RLE se vería así: *RRRRRRRVVV*.

Suponiendo que se empieza en la columna 0, entonces la posición inicial para el color rojo sería 0, mientras que la final sería 6; la inicial para el verde sería 7, mientras su final sería 9, y así sucesivamente. El campo *Renglón* se refiere al renglón que se está procesando; ***Tamaño*** es la diferencia entre la posición final y la inicial, lo cual da el tamaño de la corrida. ***Identificador*** es el número consecutivo del nodo. Mientras haya pixeles consecutivos del mismo color, el nodo se va actualizado en forma acorde; una vez que esto se rompe (debido a que se encontró un pixel que ya no pertenece a su clase), el nodo se guarda, y se da paso a un nuevo nodo vacío (campo ***Enlace al siguiente nodo***)para este color para la siguiente corrida. Por otro lado, se comienza a llenar el nodo que estará almacenando la corrida actual para el nuevo color que se acaba de encontrar. Este procedimiento se repite hasta que se termina de rastrear toda la imagen, como lo muestra el siguiente pseudocódigo.

```
r = inicio del RLE
MIENTRAS r.tag != 0
  n = siguiente RLE para el cual n.y != r.y
    SI n.y == r.y
      PARA r = cada RLE desde r hasta n
        BUSCAR( r contra el RLE que empiezan en n )
    r += 1
```

3.5. Descripción

Una vez que todos los pixeles de la imagen se han clasificado y agrupado en RLE's, lo siguiente es comenzar la formación de figuras, aún sin significado. Estas figuras serán grandes grupos de pixeles, que si cumplen ciertas condiciones se les dará el nombre de *parches*; si no las cumplen podrían ser consideradas ruido, y por lo tanto desecharse.

3.5.1. Formación de regiones en crecimiento

Algoritmos como *K-means* sólo dan como resultado centroides para las clases, y aunque ello es de utilidad para conocer la posición del parche, no da información discriminante entre objetos válidos y ruido, por ello es necesario recurrir a algún algoritmo alternativo que provea otro grado de libertad para la clasificación. Este otro factor discriminante es el área de los objetos. Si el área es mayor que un umbral establecido, entonces se toma como un objeto

válido; en caso contrario se deshecha. El algoritmo de regiones en crecimiento obtiene el área a partir de los RLE's de la etapa anterior.

Las regiones en crecimiento, o BLOBs, son agrupaciones de corridas RLE del mismo color que son vecinas entre sí. Para considerar una vecindad, un RLE debe estar conectado con otro RLE. A esta conexión se le conoce como *four-connected* (4C), Fig(3.13), o *eight-connected* (8C), como en la Fig(3.14).

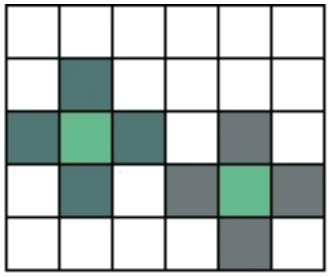

Fig. 3.13: En la región conectada en 4C sólo se toman como vecinos del pixel central (pixel claro en la figura) a los pixeles que están arriba, abajo, izquierda y derecha (pixeles obscuros)

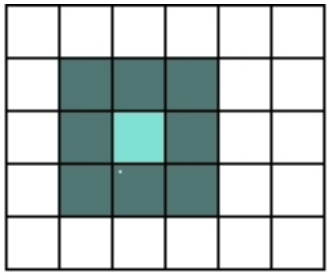

Fig. 3.14: En la región conectada en 8C se toman como vecinos del pixel central (pixel claro en la figura) a los pixeles que están arriba,abajo, izquierda, derecha y diagonales (pixeles obscuros)

Para crear un BLOB se requieren listas enlazadas que almacenen sus caracterísiticas. Éstas son guardadas en los siguientes campos

- Área
- Color

- Punto de inicio y punto de término
- Centroide
- Identificador
- Enlace al siguiente nodo

El campo ***Área*** almacena el área del BLOB en formación. *Color* contiene el color del blob. ***Punto de inicio y punto de término*** son las coordenadas de inicio y fin del blob. Los blobs se van formando de manera rectangular, por lo que dos coordenadas son suficientes para describirlo. ***Centroide*** es su centroide, el cual se utilizará más adelante a la hora de reconocer al robot. *Identificador* es el número consecutivo del blob. *Enlace al siguiente nodo* apunta al siguiente nodo.

Para que un blob se vaya formando se requiere examinar las lista enlazada de RLE's para un color en particular. El algoritmo busca en los renglones adyacentes y une las corridas que son vecinas en una conectividad 8C. Cada corrida es examinada una a la vez y comparada con las corridas de la siguiente fila. Las filas que son 8C en la fila actual y del mismo color son parte del mismo blob, y estas regiones se van formando como muestra el algoritmo. Si ambas corridas tienen sus apuntadores a nulo entonces se crea un nuevo blob y se hace una referencia de ambos a este. Si únicamente uno de estas corridas tiene un apuntador a blob no nulo entonces la corrida con el apuntador a nulo se pone al valor del blob válido, y sus parámetros son actualizados. El último caso es cuando ambos blobs ya existen, entonces se borra el más nuevo y se actualiza al más viejo. El pseudocódigo siguiente muestra el procedimiento. Lo mismo se repite para todas las clases C, y cada blob encontrado se guarda en un arreglo de blobs para su posterior discriminación.

```
r es el elemento a ser comparado con la siguiente fila
n es primer elemento de la siguiente fila tal que  n.y = r.y+1
filaab = n
MIENTRAS filaab = r.y+1
  SI r y filaab se traslapan en 8-conectividad
    SI filaab.tag > r.tag
    {
      SI blobTag == NULL
        CREAR un blob y empatarlo con r
      MEZCLAR filaab.blobTag con r.blobTag
      filaab.blobTag = r.blobTag
```

```
  filaab.tag = r.tag
}
SINO
{
    SI filaab.blobTag == NULL
      CREAR un blob y empatarlo con filaab
    MEZCLAR filaab.blobTag con r.blobTag
    filaab.blobTag = r.blobTag
    filaab.tag = r.tag
}
filaab = filaab+1
```

La Fig(3.15) muestra algunas configuraciones de blobs según se formaron para diferentes colores. Como se aprecia, son siempre regiones rectangulares cuyas dimensiones están dadas por los pixeles que se encuentran en los extremos. En caso de utilizar 4C algunos blobs se hubieran roto dando lugar a tener blobs sueltos para el mismo objeto en la imagen. 8C permite que éstos no se rompan y mantengan la forma original del objeto.

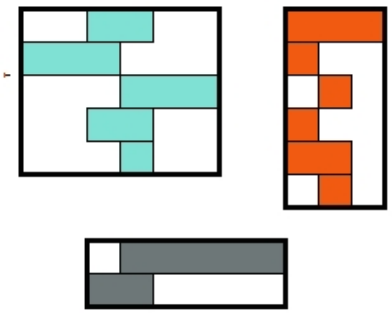

Fig. 3.15: Ejemplo de formación de los blobs

Luego de que fueron recorridas todas las listas enlazadas correspondientes a los RLEs, significa que ya la imagen fue rastreada en su totalidad, por lo que en el arreglo donde se guardaron los blobs ya existe información suficiente para clasificarlos como blobs válidos o no válidos. Esta clasificación se hace en base a cuán grande es el área del blob. La aplicación contiene un control, Fig(), que establece el área mínima para que un blob pueda considerarse válido.

3.5.2. Detección de objetos

Después de que se discriminaron los blobs más pequeños comienza la fase de detección de objetos. Se puede dar por supuesto que los blobs remanentes sean considerados como objetos, y como objetos se tienen los parches y la pelota. En un momento dado todos los objetos están distribuidos a lo largo y ancho de la cancha, por lo que es necesario comenzar a agruparlos. De acuerdo a las reglas de la Robocup, cada equipo debe estar identificado con un parche central, ya sea azul o amarillo. Cada robot de un mismo equipo debe portar un color, mientras que el otro equipo portará el color restante. Entonces el primer paso es decirle al programa qué color de parche va a utilizar nuestro equipo, y para ello hay dos botones de opción, Fig(3.16).

Fig. 3.16: Control que define el color asignado al equipo local

Una vez que se eligió el color de parche central, C_p, para nuestro equipo, lo siguiente es buscar en la lista enlazada de blobs todos aquellos que estén marcados con C_p y se guardan en un arreglo. Por lo general debería haber tantos parches C_p como robots haya en escena. Luego, para cada parche encontrado, se calcula la distancia entre este y todos los demás parches (que no sean azules ni amarillos ni la pelota) y de entre todos ellos se discrimina a aquellos que estén fuera de un radio dado R_p, por lo que sólo se toman en cuenta los 3 parches más cercanos como en la Fig(3.17). La aplicación tiene un control que define R_p, Fig(3.18).

Detección de la pelota

Para detectar la pelota el procedimiento es similar, la diferencia es que sólo hay un objeto de color naranja, el cual, una vez detectado, se envía a la máquina encargada de la IA.

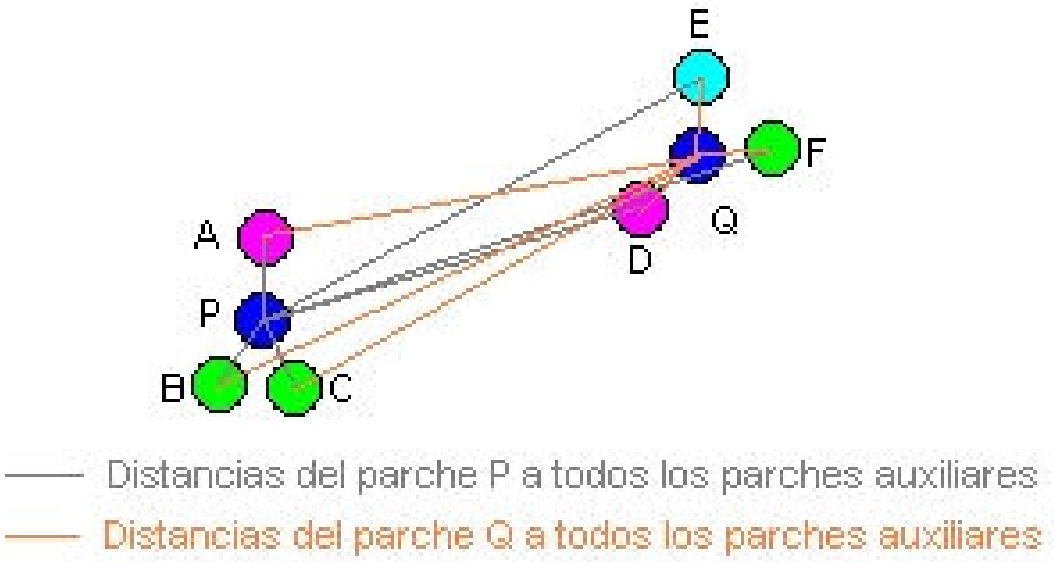

Fig. 3.17: Distancias más cercanas a los parches centrales. En color gris se tiene que los parches más cercanos a P son A, B, C, mientras que los parches más cercanos al parche Q, en naranja, son D, E, F

Detección del equipo contrario

Dado que el equipo contrario puede tener un método muy diferente para reconocer a sus robots, no tiene sentido buscar más allá del parche central; además, tampoco se sabe de antemano cuántos robots participarán, por lo que una vez que se ha detectado el parche central, su información es envíada a la máquina encargada de la IA.

Fig. 3.18: Control para establecer la distancia máxima para que un parche sea considerado como parte del objeto

3.6. Interpretación

3.6.1. Detección e identificación de los robots

Con la información obtenida hasta el momento ya es posible identificar a cada robot, y en la siguiente sección se detalla la forma de obtener la dirección.

Para identificar a cada robot, cada parche (verde, azul claro o rosa) se le dió un peso con el objeto de facilitar la identificación. Es decir, no importa el órden de colocación de los parches en el robot, sino la combinación de colores. Con 3 parches y 3 colores se tienen 27 combinaciones (3^3) diferentes, pero dado que no se toma en cuenta la repetición sólo quedan 7 opciones posibles, suficientes ya que lo máximo son 5 robots. La ponderación está dada por

```
/* (v)erde = 10, (r)osa = 100, (a)zul claro = 1000 */
#define r0 120 /* vvr */
#define r1 1020 /* vva */
#define r2 1110 /* vra */
#define r3 220 /* rrv */
#define r4 1200 /* rra */
#define r5 2010 /* aav */
#define r6 2100 /* aar */
```

3.6.2. Detección de la dirección

Para calcular la dirección hacia la que está viendo el robot se requiere conocer la posición de los parches, independientemente de su color. Los parches están colocados en forma de triángulo isóceles, con dos parches cercanos y uno más alejado, como en la Fig(3.4). Primero se crea un arreglo con la posición de los 3 parches (el central no se usa en esta etapa). Luego este mismo arreglo se ordena de tal forma que primero quede el parche más alejado, y luego los dos parches restantes. Después se obtiene el punto medio de los parches más cercanos entre sí y se guarda. Enseguida se traza una línea recta que va desde el parche más alejado hasta el punto medio recién obtenido, y utilizando el parche central se calcula el ángulo θ, como lo muestra la Fig(3.19).

3.7. Comunicaciones

3.7.1. Formato de datos

Inmediatamente después de que se obtuvo la información respectiva para los robots de nuestro equipo, ésta debe ser enviada a la máquina encargada

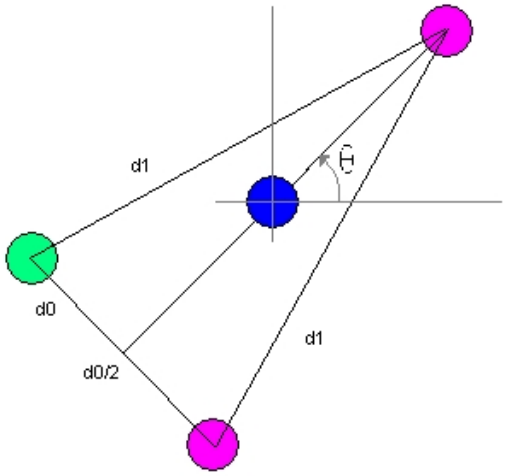

Fig. 3.19: Cálculo del ángulo θ. d_0 es la distancia más corta, mientras que d_1 representa la distancia más larga. El ángulo se obtiene de la recta que pasa por el parche central y está dirigida hacia el parche más alejado

de la IA. Sin embargo, dicha máquina debe conocer qué tipo de información está recibiendo, ya que además de la información del equipo local, también recibe la del equipo contrario y la pelota, por lo que debe tener un método para clasificar los datos que recibe.

Para ello se creó un protocolo de comunicaciones de alto nivel que le permite al sistema de IA discriminar entre los tres tipos de datos que está recibiendo. El protocolo no es nada sofisticado, ya que se está trabajando en tiempo real, y la responsabilidad de que los paquetes lleguen a su destino recayó en el protocolo TPC. El protocolo envía tramas, una por cada objeto reconocido en escena. En general el protocolo es de esta manera

```
INFO EQUIPO ID X Y VEL ANG END
```

donde

- INFO Etiqueta que indica que está comenzando una trama
- EQUIPO Indica si se trata del equipo local (P), del equipo contrario (A), o la pelota (B)
- ID Identificador del robot
- X, Y Coordenadas del robot (m)

- VEL Velocidad del robot (PIX/SEG)
- ANG Ángulo hacia el que está viendo el robot (grados)
- END Etiqueta que indica que el término de la trama

No todos los campos se aplican a todos los objetos. Para el equipo local todos los campos son válidos, mientras que para el equipo contrario se tienen los siguientes

```
INFO EQUIPO X Y VEL END
```

y para la pelota

```
INFO EQUIPO X Y VEL END
```

3.7.2. Sockets de internet

Como se ha venido comentando, la transmisión de datos se hace vía sockets de internet, lo cual ya fue explicado en le capítulo anterior. La aplicación desarrollada cuenta con controles que permiten establecer la dirección IP de la máquina de IA, así como el puerto y el número de cuadros que se procesarán por segundo, según lo muestra la Fig(3.20).

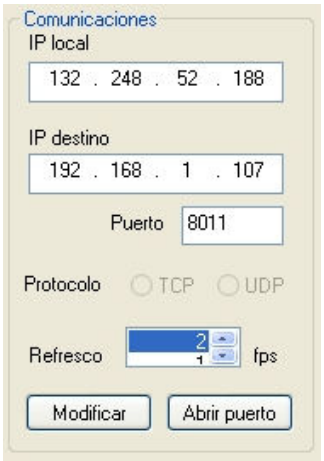

Fig. 3.20: Controles que definen el comportamiento de las comunicaciones

3.7.3. Persistencia de datos

La aplicación puede guardar el perfil de la sesión de trabajo, es decir, guarda en un archivo el estado de la aplicación en un momento dado, para que más adelante se pueda cargar dicho perfil y no sea necesario ajustar todos los controles cada vez que se arranca la aplicación. La Fig(3.21) muestra la sección donde se encuentran los botones de persistencia.

Fig. 3.21: Controles que permiten salvar y cargar sesiones

4. PRUEBAS

En este capítulo se desarrollan las pruebas hechas al sistema. Son cuatro las pruebas realizadas que tienen que ver propiamente con 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 pruebas tienen la finalidad de conocer el porcentaje de error que el sistema comete al tratar de reconocer todos los objetos en la escena. Estos errores son influenciados tanto por los dispositivos de captura, los algoritmos utilizados, así como por eventos que están fuera de control, p. ej. la iluminación. El sistema desarrollado pretende ser robusto a este tipo de errores, ya sea minimizándolos o adaptándose a ellos, como p. ej. el hecho de haber incluído controles que modifican las variables de control de la segmentación del color en tiempo real. Sin embargo, no es posible deshacerse de los errores, por lo que siempre se trabaja para que aparezcan lo menos posible, o sus consecuencias no modifiquen de manera drástica el comportamiento del sistema.

Las pruebas se harán con dos cámaras: una webcam (Philips ToUCam Pro)(CAM1), que es con la que se ha venido trabajando a lo largo del proyecto, y con una cámara digital (Sony DCR-TRV380)(CAM2) de reciente adquisición, con el objeto de mostrar el comportamiento del sistema ante diversos tipos de dispositivos de captura.

Los resultados de las pruebas son la diferencia entre la medida real (hecha con regla) del objeto sobre el campo de juego y la medida dada por la cámara. Entre más pequeño sea el porcentage de error mejor será la efectividad del sistema.

Son tres los tipos de objetos que se pueden encontrar en escena: la pelota, los jugadores contrarios y los jugadores propios. La identificación y las coordenadas de cada uno de ellos es lo que requiere el sistema de AI para poder desarrollar las estrategias, por lo que la correcta clasificación es de suma importancia.

Las pruebas comenzarán con la pelota, luego con el equipo propio, después con el equipo contrario, y finalmente la velocidad de procesamiento. Para las tres primeras hay dos tipos, a) estática para medir la precisión de las coordenandas, y b) dinámica para medir cuántos paquetes recibe el sistema de AI.

El formato de las pruebas estáticas será similar para cada uno de los objetos:

1. El objeto se coloca en escena
2. Se anota la posición real.
3. Se anota la información recolectada por el sistema. Para el caso de los robots propios hay un campo extra que es la la orientación. Se medirá ésta y luego se comparará con lo que el sistema de AI recibió.

El formato de las pruebas dinámicas es:

1. Se coloca al objeto en escena
2. El objeto deberá moverse según sea

   Pelota: se mueve la pelota dentro del campo en forma aleatoria con algún objeto que no obstruya la visibilidad ni se confunda con la pelota

   Robot: se controla al robot en forma remota y se lo mueve en forma aleatoria dentro del campo. Nota: debido a las características de construcción del robot y la recién implementación de los algoritmos de IA a la fecha de redacción de este documento, no es posible instruir al robot para que siga una línea recta, lo cual hubiera sido deseable para la realización de algunas otras pruebas.

La información enviada es recabada por el sistema de AI y se cuentan los paquetes que se hayan recibido. Del total de estos paquetes se obtendrá el número de veces que el sistema perdió al objeto, y a partir de ahí se calcula el porcentaje de error de pérdida.

4.2. Identificación de la pelota

Como se mencionó, 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 que el de dichos parches.

4.2.1. Prueba estática

Estos son los resultados del error entre la posición exacta de la pelota y la recibida por el sistema de AI para la CAM1

Coordenadas reales [cm]	Coordenadas recibidas [cm]	% de error
(110.0,101.0)	(110.6,100.7)	0.12
(50.0,31.0)	(48.6,31.1)	1.24
(130.0,70.0)	(132.6,68.8)	0.14
(10.0,100.0)	(9.3,100.1)	3.52
(21.0,14.0)	(18.1,12.9)	11.07

y para la CAM2

Coordenadas reales [cm]	Coordenadas recibidas [cm]	% de error
(70.6,70.23)	(75.0,74.5)	4.41
(149.5,134.4)	(149.0,133.3)	0.58
(48.8,137.5)	(54.5,135.0)	6.16
(109.7,17.2)	(112.0,21.5)	11.03
(184.6,81.1)	(185.0,83.0)	1.25

4.2.2. Prueba dinámica

Esta prueba determina el número de veces que el sistema de visión pierde a la pelota. El sistema de AI se alimentará con esta información y se calculará el número de paquetes perdidos.

La tabla siguiente muestra los resultados obtenidos para la CAM1

Paquetes enviados	Paquetes recibidos	% de error
2369	2349	0.84
2372	2351	0.89
2864	2855	0.31
2846	2832	0.49

y para la CAM2

Paquetes enviados	Paquetes recibidos	% de error
2191	1962	10.45

En las pruebas dinámicas para la CAM2 sólo se ha indicado un resultado dado que los datos arrojados son muy similares, por lo que se creyó conveniente no anotarlos.

4.3. Identificación de los robots propios

Como se mencionó anteriormente, en el caso de nuestros robots se debe obtener, además, la orientación del robot, para lo cual se utilizan los parches extras.

4.3.1. Prueba estática

Estos son los resultados del error entre la posición exacta del robot y la recibida por el sistema de AI para la CAM1

Coordenadas reales [cm]	Coordenadas recibidas [cm]	% de error
(82.0,74.0,207.0)	(80,74.7,214.85)	2.36
(50.0,50.0,90.0)	(47.12,47.12,89.9)	4.11
(100.0,46.0,125.0)	(102.21,43.7,126.87)	2.97
(30.0,55.0,44.0)	(24.55,55.53,41.18)	10.0
(16.5,58.0,90.0)	(9.95,58.8,89.99)	22.4

y para la CAM2

Coordenadas reales [cm]	Coordenadas recibidas [cm]	% de error
(92.5,77.1,0.0)	(95.5,79,0.0)	1.85
(173.7,100.2,1.57)	(173,102,1.57)	0.72
(53.4,133.9,2.62)	(58,132,2.35)	6.95
(46,15.9,3.14)	(49,19.5,3.14)	8.19
(51.9,52.3,4.77)	(57,55,4.71)	5.04

4.3.2. Prueba dinámica

Esta prueba pretende medir el número de veces que el sistema de AI deja de detectar a los robots propios. Si un robot no es identificado entonces no se considera válido y no es enviado al sistema de AI.

Los resultados obtenidos para la CAM1 son

Paquetes enviados	Paquetes recibidos	% de error
5268	5142	2.39
2478	2455	0.93
3178	3124	1.7

y para la CAM2

Paquetes enviados	Paquetes recibidos	% de error
4565	4432	2.91

4.4. Localización de los robots contrarios

En esta prueba se pretende detectar la posición de los robots contrarios basándose únicamente el parche central, debido a que cada equipo utiliza formas diferentes para identificar a sus robots y obtener su posición y orientación. Por lo tanto la prueba sólo consiste en obtener la posición del robot en el campo en forma dinámica.

Los resultados obtenidos con la CAM1 son

Paquetes enviados	Paquetes recibidos	% de error
2793	2771	0.79
660	650	1.52

y con la CAM2

Paquetes enviados	Paquetes recibidos	% de error
4191	4035	3.72

4.5. Velocidad de procesamiento

La máxima velocidad que alcanza el sistema es de 10 cps (cuadros por segundo). Aunque el software tiene controles que permiten modificar esta velocidad hasta los 30 cps, el rendimiento sólo llega hasta lo mencionado. Y entre más alta sea la resolución de la cámara mayor es el tiempo de procesamiento,

de ahí que para la utilización de la CAM2 hubo que realizar un subsampleo de la imagen, ya que de procesarla completa el sistema se volvía excesivamente lento. Otro punto influyente en la velocidad es el tipo de comunicación entre la cámara y la computadora. Para el caso de la webcam su comunicación es por USB, mientras que para la cámara digital es por Fire-Wire, la cual es más rápida.

5. CONCLUSIONES

A lo largo de los capítulos anteriores se hizo evidente que un proyecto de esta magnitud envuelve diferentes aspectos de la ingeniería, y ello desemboca en un sistema que da los resultados propuestos al inicio de este trabajo.

La parte más crítica es la adquisición de datos, ya que de ello dependen mucho los resultados finales. En un principio, y por cuestiones de infraestructura, el proyecto se inició utilizando webcams y basado en las reglas que en ese momento estaban vigentes. Al pasar el tiempo las reglas cambiaron así como que se adquirieron cámaras más sofisticadas, sin embargo se mantuvo el diseño original del proyecto hasta la finalización de este documento. Una webcam convencional tiene una resolución máxima de 640*480 pixeles, mientras que una cámara digital alcanza por lo menos una resolución de 1024*960 pixeles, por lo que hubo que hacerle ajustes al programa final para que aceptara cualquier tipo de resolución. A mayor resolución mejores resultados, sin embargo el tiempo de procesamiento aumenta considerablemente.

Otra etapa crítica es la de la clasificación de los pixeles, la cual puede ser hecha de muy diversas maneras, sin embargo también entra en juego el paradigma tiempo de procesamiento versus resultados, por lo cual no siempre es posible aplicar las técnicas que dan mejores resultados ya que éstas no están regularmente diseñadas para sistemas de tiempo real.

Con las técnicas utilizadas se alcanzaron los objetivos planteados al principio del proyecto los cuales eran detectar a los robots propios y contrarios y transmitir su posición. Se intentaron dos métodos diferentes para la segmentación, de los cuales el primero fue a modo experimental (K-means), y dado que en la primera etapa no dió los resultados esperados, se decidió no utilizarlo y los resultados no fueron documentados. Por otro lado se tiene también que en una primera aproximación se utilizaron 4 parches auxiliares para la identificación de los robots, imitando el trabajo de [2]. Sin embargo, tiempo después, se vió que con sólo 3 parches auxilares se podía realizar la misma tarea de identificación y orientación, por lo cual se descartó el uso de los 4 parches. En general esos fueron los cambios más trascendentales que se le hicieron al proyecto mientras duró el desarrollo de este.

Un punto a recalcar es que no fue necesario realizar ningún tipo de correc-

ción geométrica debido a que, tanto por la altura como por las características de las lentes de las webcams, la distorsión que se presentó fue mínima y no afectó las mediciones. Sin embargo, con las nuevas cámaras y las lentes de gran angular, y también por las nuevas reglas, la altura de las cámaras aumentó, por lo que ya fue estrictamente requerido el uso de lentes de gran angular, con la consecuente distorsión correspondiente; la corrección no fue tema de este trabajo, pero más adelante se plantea como una de las futuras mejoras al sistema.

De los datos mostrados puede verse la tendencia que toman los errores. Ello es debido principalmente a que la cancha no está iluminada uniformemente, por lo que en ciertas zonas habrá diferencias notables que dificultan el reconocimiento de los objetos en la escena. Particularmente esto se nota más en las orillas, mientras que en el centro, por debajo de las cámaras, es donde el error se hace mínimo. También para el caso de la CAM2 se tiene que ya existe una muy notable distorsión geométrica debido a la lente de gran angular. Más adelante se hablará de algunas propuestas para mejorar el rendimiento del sistema.

5.1. Pelota

De los resultados derivados de las pruebas a la pelota se ve que los errores son pequeños; y aunque ello es alentador no hay que perder de vista que la pelota es el objeto que además de ser el más pequeño en escena para reconocer, es el más dinámico, por lo que en ocasiones se podría llegar a dificultar su ubicación ya dentro de un ambiente de competencia. Sin embargo los resultados demuestran que la confiabilidad del sistema para su reconocimiento es bastante aceptable.

5.2. Robots propios

La localización de los robots propios arrojó resultados buenos. Pero ha de notarse las condiciones con las que fueron realizadas las pruebas porque el sólo cambio de cámara afecta de manera notable el comportamiento del sistema (y una situación curiosa es la de que utilizando una cámara de mayor calidad y resolución, donde se supondría que el resultado debe ser mejor, no sucedió así, los mejores resultados fueron obtenidos con una webcam en particular, que fue la que se utilizó de manera recurrente durante el desarrollo del proyecto). Sin embargo, como se ha venido mencionando, las limitantes de ese tipo de cámaras ya no permiten su uso con las reglas actuales; por otro lado se tiene

que para cumplir con las nuevas reglas, entre otras cosas, se requiere el uso de lentes de grandes angulares, con las consiguientes situaciones por resolver, p.ej., la distorsión geométrica de las lentes.

En cuanto al error de posición este es pequeño, y dado que el sistema es de lazo cerrado, se tiene entonces que dicho error se va corrigiendo automáticamente. Cabe mencionar además que, a la fecha de redacción de este trabajo, el robot introduce un ligero error en su movimiento, pero por la misma situación del lazo cerrado su influencia es mínima.

Como se mencionó al final del capítulo anterior, los errores son más notables cuanto más se aleja el robot o la pelota del centro, y ello es debido en parte por la iluminación que no es constante a lo largo de la cancha (iluminación centralizada), y en parte por la distorsión geométrica, que en mayor o menor grado (dependiendo de la cámara) se hace más notorio en la periferia.

Otro punto a recalcar son las variaciones de intensidad luminosa a lo largo y ancho del campo de juego, la cual depende del material del que este hecho, del número y tipo de lámparas y su colocación, y todas estas variables son altamente volátiles y dependen del lugar de la competencia. Dentro del laboratorio estas variables están controladas, o al menos son constantes, por lo que el sistema se desarrolló sobre ellas, pero dentro de una competencia se deben solventar problemas que no se tenían en el laboratorio, y que en muchas ocasiones se resuelven in-situ, soluciones que posteriormente se le integran a la aplicación.

5.3. Velocidad de procesamiento

En el capítulo anterior se mencionaron 2 elementos que influyen en la velocidad. Por un lado la resolución de la cámara y por otro el tipo de comunicación entre esta y la computadora. Cabe mencionar que existen otros directamente relacionados con el software de captura y los algoritmos de procesamiento. Un posible cuello de botella es la conversión de RGB a HSI debido a la no linealidad de éste último. También se tiene el uso extensivo de listas enlazadas, y por consecuencia, un número masivo de llamadas a las funciones malloc() y free(). En la siguiente sección se plantean algunas posibles soluciones para mejorar el rendimiento del sistema.

5.4. Futuro

Dado que cualquier sistema es perfectible, existen ciertos renglones dentro de este proyecto donde se pueden realizar modificaciones e integrar mejoras

para obtener mejores resultados, ya que además, es esta la primera versión de este sistema de visión.

- En primera instancia se tiene que hay que integrarle la corrección geométrica que minimice la aberración introducida por la lente de gran angular.

- Actualmente se debe de abrir una instancia de la aplicación por cada cámara que se utilice, y por consecuencia ajustar los parámetros independientemente. Otra mejora sería fusionar dos cámaras en la misma instancia y de esta manera tener la imagen completa del campo y poder trabajar con ella de una sola vez, ahorrando tiempo y homogeneizando los parámetros de control y calibración.

- Experimentar con otros espacios de color, p. ej. YUV. El espacio RGB se utilizó en un principio, pero dado que si la iluminación cambia entonces todas las componentes cambian dificultaba en extremo la nueva recalibración, de ahí que se haya decidido utilizar el espacio HSI, asunto que ya fue tratado en detalle.

- Implementar técnicas de segmentación más sofisticadas, p. ej. probabilísticas, y técnicas de localización de objetos, p. ej., flujo óptico, que aunque fueron abordadas en este trabajo (segmentación bayesiana), por cuestión de tiempo de ejecución no se integró al sistema.

- Introducir hilos de ejecución para utilizar el paralelismo y acelerar el tiempo de ejecución, es decir, mientras un cuadro se está capturando procesar el anterior, lo que daría pie tanto a procesar más cuadros por segundo, como implementar otros algoritmos, como se ha propuesto más arriba.

- Aunque la función de coseno inverso para la transformación de RGB a HSI está implementada por tabla, el hecho de que sea circular la componente H (hue o matiz) implica agregar una comprobación más por cada pixel, lo que al final representa más tiempo consumido. Se plantea entonces que no sólo para la componente H, sino para S e I, se haga algún tipo de búsqueda por tabla. El sistema en su totalidad no está pensado para correr en computadoras modestas, por lo que no habría problema en precargar tablas de datos medianamente grandes, lo cual podría ayduar a mejorar el rendimiento.

- Hablando en general del proyecto completo se pueden hacer algunas recomendaciones tales como

  Permitir poder variar la velocidad de los motores del robot (actualmente es una velocidad fija)

  Integrar un control PID básico en el robot

  Implementar una IA con movimientos básicos

  Desarrollar un simulador

  Simplificar en general la preparación del sistema para competencia o demostración, ya que es complicado echar a andar cada uno de los elementos, y algunas veces aparecen problemas como falta de conectividad wireless o de conexiones eléctricas.

BIBLIOGRAFíA

[1] http://www.robocup.org/Intro.htm

[2] *Martínez, L. A.*, **Sistema de visión para el equipo de robots autónomos del ITAM**, Tesis ITAM, México, 2004

[3] http://www-2.cs.cmu.edu/čoral/publications/class_rescat.html

[4] http://robocup.mi.fu-berlin.de/publications.html

[5] *Coiffet, P.*, **Robot Technology**, Vol. 1, Prentice Hall, USA, 1983

[6] *Fu, K. S., Gonzalez, R. C., Lee, C. S.*, **ROBOTICS: Control, Sensing, Vision and Intelligence**, Mc. Graw Hill, USA, 1987

[7] *Ballard, D. H, Brown, M. B.*, **Computer vision**, Prentice Hall, USA, 1982

[8] *Castleman, K. R.*, **Digital Image Processing**, Prentice Hall, USA, 1996

[9] *Davies, E.R.*, **Machine vision**, Academic Press, 1990, London

[10] *Schaarschmidt, W.*, **Reconocimiento de gestos clave para interactuar con un robot móvil**, Tesis de Maestría, FI, UNAM, México, 2004

[11] www.es.wikipedia.org/wiki/Robot

[12] *Urenda, I.*, **Manipulación de imágenes bitmap (y II)**, *Revista "Sólo Programadores"*, Pág. 32-37, No. 84, Año VIII, 2da época, España, 2001

[13] *Leon-Garcia, A.*, **Probability and Random Processes for Electrical Engineering**, Addison-Wesley, USA, 1994

[14] *Gonzalez, R.*, **Digital Image Processing**, 2nd Ed, Addison-Wesley, USA, 1987

[15] *Salomon, D.*, **Data Compression, The Complete Reference**, Springer-Verlag, USA, 1988