VNIVERSIDAD NACIONAL AVTNOMA DE MEXICO

UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

PROGRAMA DE MAESTRIA Y DOCTORADO EN INGENIERIA

FACULTAD DE INGENIERIA

SISTEMA DE NAVEGACIÓN PARA ROBOTS MÓVILES UTILIZANDO UNA ARQUITECTURA DELIBERATIVA/REACTIVA

T E S I S

QUE PARA OPTAR POR EL GRADO DE:

MAESTRO EN INGENIERIA

ELECTRICA - SISTEMAS ELECTRONICOS

P R E S E N T A :

DAVID CORTES POZA

TUTOR:

DR. JESUS SAVAGE CARMONA

2007

JURADO ASIGNADO:

Presidente:	Dr. Fernando Arámbula Cosío
Secretario:	Dr. Marco Arteaga Pérez
Vocal:	Dr. Jesús Savage Carmona
$1^{er.}$ Suplente:	Dra. Ana Lilia Laureano Cruces
$2^{do.}$ Suplente:	Dr. Pablo Roberto Pérez Alcázar

Lugar o lugares donde se realizó la tesis:

Laboratorio de Biorrobótica.
Edificio de Posgrado Bernardo Quintana, $2^{do.}$ piso
Facultad de Ingeniería, UNAM

TUTOR DE TESIS:

Dr. Jesús Savage Carmona

FIRMA

a Elisa, Emilio
Aída, Yuriria y Rogelio

a Úrsula

Agradecimientos

Al Dr. Jesús Savage Carmona
por su apoyo y asesoría.

A mis sinodales:
Dr. Marco Arteaga Pérez,
Dr. Fernando Arámbula Cosío,
Dra. Ana Lilia Laureano Cruces,
Dr. Pablo Roberto Pérez Alcázar
por la revisión de la tesis.

A CONACyT por
la beca otorgada.

A la UNAM.

Índice general

Índice de figuras

Resumen

En este trabajo se desarrollaron un sistema de visión y un sistema de planeación/navegación para un equipo de robots móviles que participan en la competencia *RoboCup Small Size*.

El sistema de visión se desarrolló utilizando umbralización para la segmentación de los pixeles de la imagen uno a uno y se implementó un algoritmo de crecimiento de regiones para agrupar los pixeles en regiones por colores. Posteriormente, a partir de las regiones detectadas, se realiza el reconocimiento del robot y se calcula su orientación.

El sistema de planeación/navegación fue diseñado con una arquitectura deliberativa/reactiva (híbrida) contemplando las características del ambiente donde se desenvuelven los robots. Dicho ambiente es extremadamente dinámico, lo cual impone la necesidad de la parte reactiva; por otro lado se requiere que se pueda realizar la planeación de estrategias de juego a largo plazo, de lo cual se encarga la parte deliberativa.

La navegación se desarrolló utilizando el comportamiento de campos potenciales que corresponde a una arquitectura reactiva.

El sistema de visión resulta ser muy eficiente dado que logró tener un buen desempeño aun utilizando webcams que entregan video de baja calidad. Con el algoritmo de campos potenciales se logra que el robot evite chocar con los obstáculos y alcanzar su objetivo. El planeador se encarga de decidir a donde se debe dirigir el robot en cada instante y que objetos debe evadir.

Capítulo 1

Introducción

1.1. Antecedentes

1.1.1. *RoboCup*

La *RoboCup* es una de las principales competencias de robótica en el mundo, donde los robots diseñados, construidos o programados por diferentes equipos compiten según un conjunto determinado de reglas. La *RoboCup* es una iniciativa de investigación y educación internacional que intenta fomentar la investigación en inteligencia artificial y robótica proveyendo un problema estándar donde un amplio rango de tecnologías pueden ser integradas y examinadas. Para este propósito *RoboCup* escogió el juego de fútbol como competencia principal. Para que un equipo de robots ejecute un juego de fútbol deben resolverse numerosos problemas ligados a la percepción, mecánica, inteligencia artificial, control y electrónica. [20]

RoboCup Soccer es una tarea para un equipo de robots móviles de movimientos rápidos bajo un ambiente dinámico. Dentro de las diferentes categorías de *RoboCup Soccer* destacan [20]:

- Liga de simulación
- Liga de robots *Small Size* (tamaño pequeño)
- Liga de robots Middle Size (tamaño mediano)
- Liga de robots con cuatro patas (AIBOs)
- Liga humanoide

Uno de los mecanismos más sencillos para promocionar la investigación en ingeniería es tener una meta importante a largo plazo. Construir un robot para que juegue fútbol soccer por sí mismo no genera un impacto social importante pero la realización del proyecto sería, sin duda, el mayor éxito dentro del campo de la robótica. Al conseguir esta meta, se habrán resuelto muchos de los grandes retos que tiene la robótica.

La meta máxima planeada para *RoboCup* es:

> "Para mediados del siglo XXI un equipo completo de fútbol soccer constituido por robots humanoides autónomos, ganará un juego de fútbol cumpliendo con la reglas oficiales de la FIFA, contra el ganador de la copa del mundo más reciente".

La idea de robots jugando fútbol la tuvo el profesor Alan Mackworth (University of British Columbia, Canada) en un artículo llamado "On Seeing Robots". De manera independiente, un grupo de investigadores japoneses discutieron los posibles problemas de una competencia así. En junio de 1993 un grupo de investigadores vieron posible hacer una competencia de robótica bajo el nombre Robot J-League (J-League se refiere a la liga de fútbol soccer profesional japonesa). En menos de un mes investigadores de todo el mundo pidieron que la iniciativa se extendiera y se formalizara como un proyecto internacional, por lo cual el nombre cambió a *RoboCup*. En septiembre de 1993 se llevó a cabo la primera iniciativa pública con reglas específicas. [20]

Las primeras reglas oficiales se hicieron en 1994. Se llevan a cabo revisiones anuales tanto logísticas como técnicas. La base de la regulación actual fue realizada en 1996 y es modificada intentando promover ciencia y tecnología. Las revisiones anuales se llevan a cabo por el comité y son discutidas con los participantes e investigadores destacados en el campo para establecer las nuevas reglas [22]. Los cambios también intentan lograr un entorno más real y eliminar ambientes artificiales. [20]

1.1.2. Categoría *Small Size*

La categoría *Small Size* se enfoca en el problema de cooperación y control en multi-agentes inteligentes, en un ambiente muy dinámico, con un sistema híbrido centralizado/distribuido. [21]

Un juego de *robot soccer* se lleva a cabo entre dos equipos de cinco robots cada uno. Las reglas para esta categoría se llaman *F180* [22]. En ellas se especifica que

el robot debe caber en un cilindro de diámetro de 180mm y de alto de 15cm, a menos que se utilice visión a bordo. La cancha tiene una alfombra verde y mide 2.8m de largo por 2.3m de ancho. La pelota es una pelota de golf naranja. Cuando el equipo utiliza visión global, usan una cámara colgada por encima de la cancha y se utiliza una computadora fuera del campo de juego, para realizar las tareas de procesamiento de video y de enviar las posiciones a los robots. En general, esta computadora también es utilizada para recibir las señales del árbitro y enviarlas hacia los robots. Las comunicaciones entre esta computadora y los robots o entre robot y robot (si se utiliza), deben ser inalámbricas. [21]

1.1.3. ¿Qué es un robot?

Un robot es un dispositivo electrónico motorizado controlado por una computadora que desempeña tareas de manera autónoma, ya sea bajo 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 ensamble en líneas de manufactura, manipulación de objetos pesados o peligrosos, trabajo en el espacio, entre otras.

1.1.4. Robótica

La robótica es una ciencia o rama de la tecnología que estudia el diseño y construcción de máquinas capaces de desempeñar tareas, que en principio pueden ser realizadas por el ser humano, o que requieren el uso de inteligencia (robots). Las ciencias y tecnologías de las que deriva son: álgebra, autómatas programables, máquinas de estados, mecánica e informática.

La historia de la robótica ha estado unida a la construcción de artefactos que trataban de materializar el deseo humano de crear seres semejantes a nosotros que nos facilitaran diferentes trabajos. El ingeniero Leonardo Torres Quevedo (creador del primer mando a distancia para su torpedo automóvil y del primer transbordador aéreo) acuñó el término "automática" en relación con la teoría de la automatización de tareas tradicionalmente asociadas a los humanos.

Karel Capek acuñó el 1921 el término Robot en su obra dramática "Rossum´s Universal Robots" a partir de la palabra checa Robbota que significa servidumbre o trabajo forzado. El término robótica es acuñado por Isaac Asimov definiendo a la ciencia que estudia a los robots. Asimov creó también las Tres Leyes de la Robótica.

En la ciencia ficción el hombre ha imaginado a los robots visitando nuevos mundos, tomando el poder o simplemente liberando al hombre de sus tareas cotidianas.

La robótica ha alcanzado un nivel de madurez elevado en los últimos tiempos y cuenta con un aparato teórico sofisticado. Sin embargo, algunas cosas que para los humanos son muy sencillas, como caminar, correr o coger un objeto sin romperlo aun requieren un poder de cálculo muy complejo para ser igualadas. Sin embargo, se espera que el contínuo aumento de la potencia de las computadoras y las investigaciones en inteligencia artificial, visión artificial y otras ciencias paralelas nos permitan acercarnos un poco más a las fantasías soñadas desde hace mucho por los pioneros en la ingeniería, y a los escenarios peligrosos que nos adelanta la ciencia ficción.

1.1.5. Visión computacional

La visión es la capacidad de percibir objetos o eventos a distancia por detección de luz (ondas electromagnéticas o fotones) los cuales han sido reflejados o emitidos por un objeto. La información de esta detección puede ser bidimensional (una imagen) o tridimensional (dos dimensiones y tiempo).

La visión artificial en robótica es el campo de la computación que estudia el uso de cámaras como sensores, donde las cámaras actúan como ojos. El principio se basa en que la luz reflejada en los objetos, al pasar a través de una lente, forma una imagen que puede ser procesada. La visión computacional es la ciencia que se encarga de programar una computadora para procesar y extraer la información contenida en las imágenes y video. Este es un problema que implica el procesamiento de señales aplicado a dos, tres o más dimensiones. Esto último es lo más destacado de las dificultades puesto que las computadoras modernas tienen un diseño en serie lo cual significa que sólo pueden procesar los datos en forma secuencial. El procesamiento en paralelo puede ser más adecuado para señales multidimensionales, como las tareas de visión y, de hecho, así es como el sistema de visión humano se encuentra organizado.

La visión artificial es también conocida como visión por computadora y constituye un subcampo de la inteligencia artificial. El propósito de la visión artificial es programar un sistema para que entienda una escena o las características de una imagen. Los objetivos típicos de la visión artificial incluyen la detección, segmentación, localización y reconocimiento de ciertos objetos en imágenes, la evaluación de resultados, registro de diferentes imágenes de una misma escena u objeto, seguimiento de un objeto en una secuencia de imágenes, mapeo de una escena para generar un

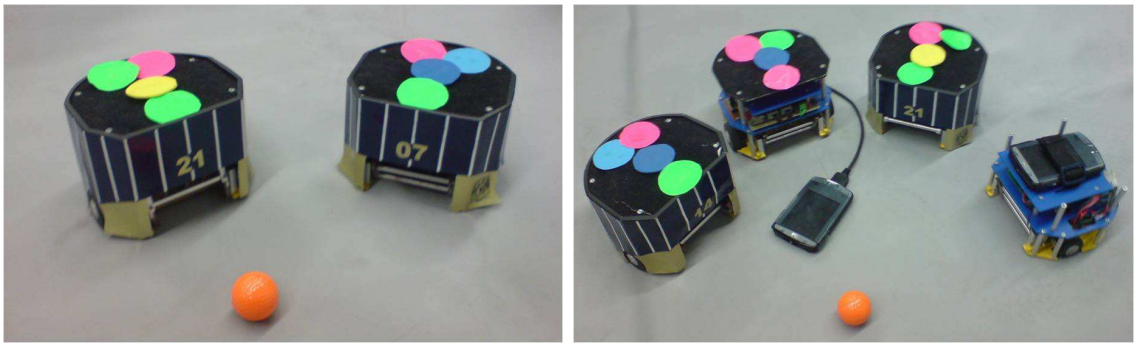

Figura 1.1: Robots del laboratorio de Biorobótica para la liga *Small Size*

modelo tridimensional de la misma, estimación de las posturas tridimensionales de los humanos, la búsqueda de imágenes digitales por su contenido, entre otros.

1.1.6. Inteligencia artificial

La inteligencia artificial se define como aquella inteligencia exhibida por artefactos creados por humanos (es decir, artificiales). A menudo se aplica a las computadoras y se puede decir que la inteligencia artificial se encarga de modelar la inteligencia humana en sistemas computacionales. El nombre también es utilizado para referirse al campo de la investigación científica que intenta acercarse a la creación de tales sistemas. Aunque la inteligencia artificial está rodeada de mitos y ciencia ficción, se trata de una rama de la computación que trata sobre comportamientos inteligentes, aprendizaje y adaptación en máquinas. La inteligencia artificial hoy en día es una de las áreas con más retos en las ciencias de la computación. Posee amplias relaciones con disciplinas matemáticas como el álgebra y la estadística, tomando de éstas algunas herramientas para desempeñar su labor.

El término "inteligencia artificial" fue inventado en 1956 en la conferencia de Darmouth. Los juegos matemáticos antiguos como el de las torres de Hanoi (aproximadamente 3000 a.c.) demuestran el interés por la búsqueda de un algoritmo iterativo que resuelva el problema, es decir, una inteligencia artificial capaz de ganar en los mínimos movimientos posibles.

1.2. Infraestructura

En el Laboratorio de Biorobótica se cuenta con cuatro robots que cumplen con las dimensiones establecidas en las reglas de la liga *Small Size* (figura 1.1).

Se cuenta también con un punto de acceso (*access point*) inalámbrico para realizar las comunicaciones entre el sistema de visión, el de planeación/navegación y los robots.

Además, será necesario utilizar dos computadoras PC con Windows XP para realizar las labores de visión e inteligencia.

1.2.1. Especificaciones de los robots

Los robots que se utilizarán para el proyecto emplean tracción diferencial: se emplean dos motores que hacen girar a dos ruedas y una tercera rueda loca, formando un triángulo entre las tres).

Los motores son controlados por una tarjeta que cuenta con tres pics. Además, cuentan con una *PocketPC* a bordo (ver figura 1.1 derecha), en donde se realizan las tareas de comunicación inalámbrica (utilizando WiFi) y en donde, en un futuro, se alojará a la inteligencia de los robots. Las *PocketPC's* están conectadas vía puerto serie con las tarjetas de los robots.

Los robots tienen un diámetro de 18cm y un alto de 15cm. En la parte superior tienen parches de colores que serán utilizados por el sistema de visión para realizar la detección.

1.3. Objetivo general

El objetivo de este trabajo es el desarrollo de un *sistema de visión* y un de *sistema de planeación/navegación (Director Técnico)* para un equipo de cinco robots móviles que competirán en la competencia *RoboCup*, en la categoría *Small Size*.

Es necesario desarrollar un sistema de lazo cerrado, en donde el sistema de visión deberá realizar la detección de los robots móviles y la pelota; el sistema de planeación/navegación recibirá las posiciones de dichos objetos y deberá emitir los comandos necesarios a los robots para que jueguen

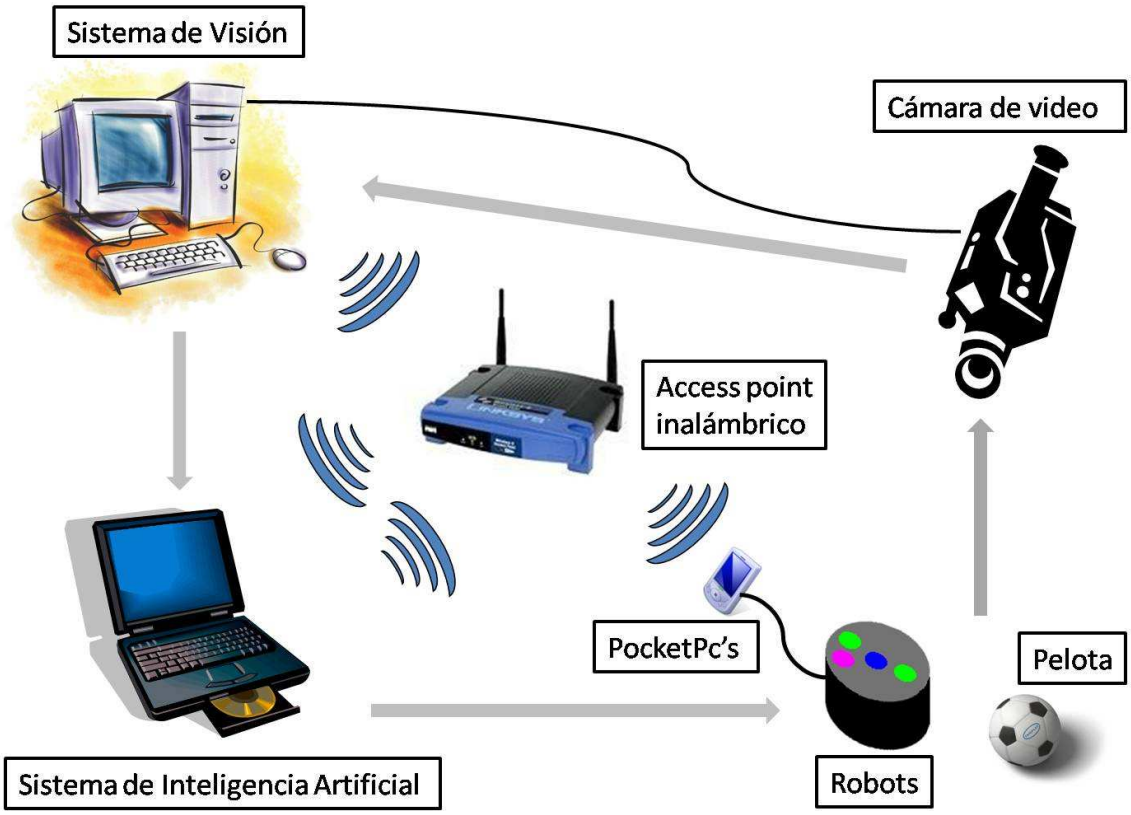

Figura 1.2: Diagrama del sistema

fútbol.

En la figura 1.2 se presenta un diagrama de bloques del sistema que se pretende desarrollar.

1.4. Objetivos particulares

Los objetivos particulares de este trabajo son los siguientes:

- Para desarrollar el sistema de visión, es necesario implementar los siguientes módulos:
  - Captura de video
  - Calibración de colores

  - Segmentación por color
  - Formación de regiones
  - Detección de robots a partir de las regiones
- El sistema de planeación/navegación utilizará *campos potenciales* para realizar los movimientos y evadir obstáculos.
- Ambos sistemas deberán contar con un módulo de comunicación para que el sistema de visión pueda enviar las posiciones encontradas al sistema de planeación/navegación.

1.5. Organización del trabajo escrito

El presente trabajo está organizado en seis capítulos y un apéndice. A continuación se hace una breve descripción de cada uno de ellos:

- En el capítulo 2 se desarrollan los aspectos teóricos de visión computacional, inteligencia artificial y todos los temas que es necesario comprender para la realización del proyecto.
- El capítulo 3 explica cómo se implementó el sistema de visión.
- El capítulo 4 explica cómo se implementó el sistema de planeación/navegación.
- En el capítulo 5 se describen las pruebas que se realizaron con el sistema ya funcional y los resultados que se obtuvieron.
- En el capítulo 6 se encuentran las conclusiones del proyecto y se plantean los retos que quedan pendientes para resolver en el futuro.
- En el apéndice se encuentra el manual de usuario para poder poner a funcionar el sistema completo: Robots, *PocketPC's*, sistema de planeación/navegación y sistema de visión.

Capítulo 2

Marco teórico

2.1. Visión computacional

2.1.1. Espacios de color

El propósito de un espacio de color es estandarizar la especificación de los colores. En esencia, un espacio de color es un sistema de coordenadas en donde cada color es representado por un solo punto. [4]

La imágenes a color pueden ser modeladas usando tres canales de datos de imágenes monocromáticas, en donde cada canal de datos corresponde a un color diferente. La información guardada en cada uno de los canales es realmente la intensidad luminosa de cada banda espectral. Cuando la imagen es desplegada, la información de intensidad que tiene cada canal es mostrada en la pantalla utilizando el color correspondiente. En general, las imágenes a color son representadas utilizando los colores rojo, verde y azul (RGB). Usando el estándar monocromático de 8 bits por pixel, una imagen a color RGB requiere 24 bits por pixel (8 bits por cada uno de los tres canales). El total de colores en una imagen de 24 bits RGB es $(2^8)^3 = 16,777,216$. [19]

En el espacio RGB, los colores son descompuestos en sus componentes espectrales primarias de rojo, verde y azul. Este modelo esta basado en un sistema de coordenadas cartesianas. El subespacio de color de interés es un cubo (figura 2.1) en el que los colores rojo, verde y azul son los tres vértices que se encuentran sobre los ejes; el negro esta en el origen y el blanco esta en el vértice mas alejado del origen, y el cian, magenta y amarillo son otros tres vértices. La escala de grises en este modelo se encuentra en la diagonal que se extiende desde el origen (negro) hasta el vértice

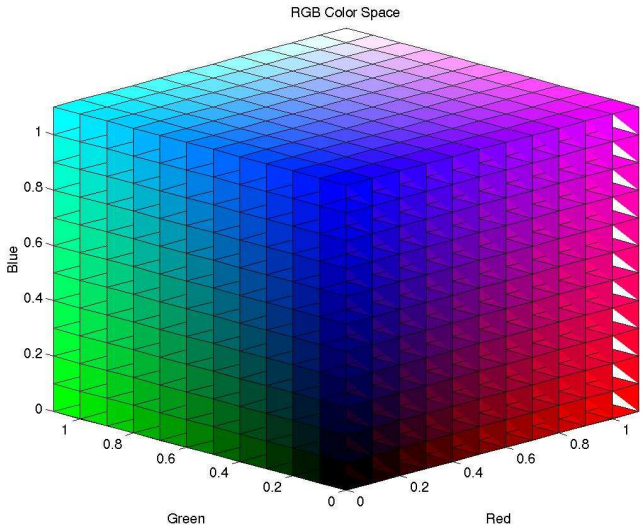

Figura 2.1: Espacio de color RGB

mas alejado (blanco). [4]

Espacio de color HSI

El modelo RGB, desafortunadamente, no ofrece una descripción práctica de los colores para la interpretación humana. Los humanos no describimos los colores por sus componentes en rojo, verde y azul, sino por su matiz, saturación e intensidad o brillantez (HSI: Hue, Saturation, Intensity). El matiz es un atributo que nosotros concebimos normalmente como color (amarillo, rojo, naranja, etc.); la saturación da una medida de qué tan "diluido" con luz blanca está el color; la intensidad es una medida del brillo del color. En el espacio de color HSI quedan separadas la información de color (matiz y saturación) de la información de intensidad. Por esta razón, resulta tan útil este modelo cuando se va a hacer segmentación de colores, en la que la información de intensidad resulta poco importante. [4]

En la figura 2.2 se muestra la representación geométrica del espacio HSI. Se observa cómo la información de intensidad queda separada de la información de color al estar contenida en el eje vertical. En cambio, la información de matiz y saturación (que definen el color) se encuentran en los cortes circulares horizontales del bicono. El matiz está dado por el ángulo azimutal y la saturación aumenta con el radio.

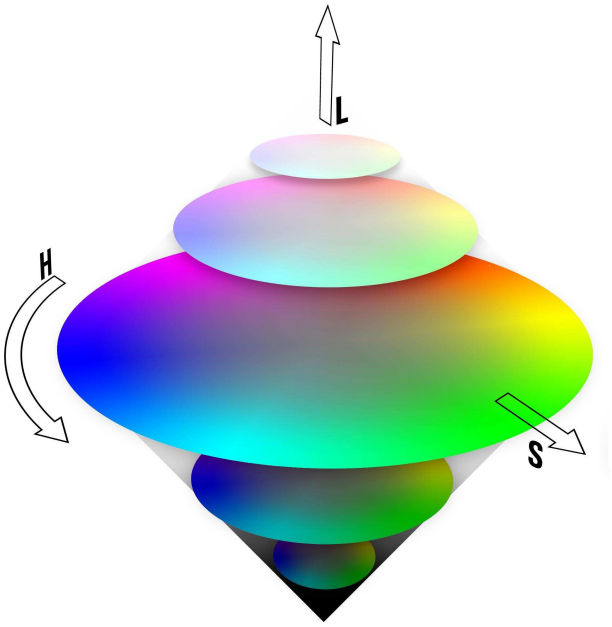

Figura 2.2: Espacio de color HSI

Si se tiene una imagen RGB y se quiere obtener sus componentes HSI o viceversa, es necesario realizar una transformación, que se describe a continuación:

Transformación de RGB a HSI

Dada una imagen en el espacio RGB, la componente H de cada pixel se obtiene utilizando la siguiente ecuación:

$$H = \begin{cases} \theta & si\, B \leq G \\ 360 - \theta & si\, B > G \end{cases} \tag{2.1}$$

con

$$\theta = cos^{-1} \left\{ \frac{\frac{1}{2}[(R-G)+(R-B)]}{[(R-G)^2+(R-B)(G-B)]^{1/2}} \right\} \tag{2.2}$$

La componente de saturación está dada por

$$S = 1 - \frac{3}{R+G+B}[min(R,G,B)]. \tag{2.3}$$

Y finalmente, la componente de intensidad se calcula de la siguiente manera:

$$I = \frac{1}{3}(R + G + B). \tag{2.4}$$

Se asume que los valores de RGB han sido normalizados al intervalo [0,1], y que el ángulo θ esta medido con respecto al eje del rojo del espacio HSI. La componente de matiz (H) puede ser normalizada entre [0,1] dividiéndola entre 360° todos los valores de la ecuación 2.1.

Estos resultados son deducidos geométricamente a partir de la representaciones geométrica de ambos espacios.

Transformación de HSI a RGB

Dados valores de HSI normalizados entre [0,1], se obtendrán las componentes RGB para dicho color. La ecuación que se aplique dependerá del valor de H, dado que hay tres sectores de interés que corresponden a la separación de 120° de los componentes primarios RGB. Para comenzar, se multiplica la componente H por 360° para llevarla nuevamente al intervalo [0°,360°].

Sector RG ($0° \leq H < 120°$): Cuando H se encuentra en este sector, las componentes RGB se obtienen con las ecuaciones

$$B = I(1 - S) \tag{2.5}$$

$$R = I\left[1 + \frac{S\cos H}{\cos(60° - H)}\right] \tag{2.6}$$

y

$$G = 1 - (R + B). \tag{2.7}$$

Sector GB ($120° \leq H < 240°$): Si el valor de H se encuentra en este intervalo, primero se le restan 120°.

$$H = H - 120°. \tag{2.8}$$

Entonces, las componentes RGB son

$$R = I(1 - S) \tag{2.9}$$

$$G = I\left[1 + \frac{S\cos H}{\cos(60^\circ - H)}\right] \tag{2.10}$$

y

$$B = 1 - (R + G). \tag{2.11}$$

Sector BR ($240^\circ \leq H \leq 360^\circ$): Finalmente, si H se encuentra en este intervalo, se le restan 240°:

$$H = H - 240^\circ. \tag{2.12}$$

Entonces, las componentes RGB son

$$G = I(1 - S) \tag{2.13}$$

$$B = I\left[1 + \frac{S\cos H}{\cos(60^\circ - H)}\right] \tag{2.14}$$

y

$$R = 1 - (G + B). \tag{2.15}$$

2.1.2. Función de distribución acumulativa y función de densidad de probabilidad

Caso contínuo

La función de distribución acumulativa (*fda*) de una variable aleatoria X, se define como la probabilidad del evento $\{X \leq x\}$ [8]:

$$F_X(x) = P[X \leq x] \quad para - \infty < x < +\infty, \tag{2.16}$$

es decir, la probabilidad de la variable aleatoria X en el intervalo $(-\infty, x]$.

La función de densidad de probabilidad de una variable aleatoria X (*fdp*), se define como la derivada de $F_X(x)$:

$$f_X(x) = \frac{dF_X(x)}{dx} \tag{2.17}$$

Ésta es una manera mas útil de presentar la información contenida que la función de distribución acumulativa.

La *fdp* representa la densidad de probabilidad en el punto x, es decir, la probabilidad de que X se encuentre en un pequeño intervalo en la vecindad de x ($x < X \leq x + \Delta x$) es:

$$\begin{aligned} P[x < X \leq x + \Delta x] &= F_X(x + \Delta x) - F_X(x) \\ &= \frac{F_X(x + \Delta x) - F_X(x)}{\Delta x}\Delta x \end{aligned} \tag{2.18}$$

Si la *fda* tiene derivada en x, entonces mientras Δx se hace pequeño,

$$P[x < X \leq x + \Delta x] \approx f_X(x)\Delta x \tag{2.19}$$

Entonces $f_X(x)$ representa la densidad de probabilidad en el punto x dado que la probabilidad de que X se encuentre en un pequeño intervalo en la vecindad de x es aproximadamente $f_X(x)\Delta x$. La derivada de la *fda*, cuando existe, es positiva dado que la *fda* siempre es una función de x no decreciente, entonces:

$$f_X(x) \geq 0 \tag{2.20}$$

En la ecuación 2.19 se encuentra otra manera de calcular las probabilidades de una variable aleatoria X. Se puede comenzar por encontrar una función no negativa $f_X(x)$, llamada función de densidad de probabilidad, que indica la probabilidade de que X esté en un pequeño intervalo de tamaño dx alrededor del punto x. Las probabilidades de eventos que involucren a X, entonces, se pueden expresar en términos de la *fdp* al sumar las probabilidades de los pequeños intervalos de tamaño dx. Dado que el tamaño de los intervalos es casi cero, se obtiene una integral en términos de la *fdp*. Por ejemplo, la probabilidad de un intervalo $[a, b]$ es [8]:

$$P[a \leq x \leq b] = \int_a^b f_X(x)dx \tag{2.21}$$

La probabilidad de un intervalo es el área bajo $f_X(x)$ en ese intervalo.

Caso discreto

Las variables aleatorias discretas, pueden tomar un valor de un conjunto finito, o a lo mucho infinito contable, de valores $S_X = x_0, x_1, ...$ La función de densidad

de probabilidad es entonces el conjunto de probabilidades $p_X(x_k) = P[X = x_k]$ para cada uno de los elementos en S_X. Esto se expresa de la siguiente manera [8]:

$$f_x(x) = \sum_k p_X(x_k)\delta(x - x_k) \tag{2.22}$$

Análogamente al caso contínuo en donde la función de distribución acumulativa se escribe como la integral de la *fdp*, en el caso discreto, la *fda* puede escribirse como la suma ponderada de funciones escalón unitarias:

$$F_X(x) = \sum_k p_X(x_k)u(x - x_k), \tag{2.23}$$

donde $p_X(x_k) = P[X = x_k]$ indica la magnitud de los saltos para cada escalón en la *fda*. [8]

2.1.3. Valor esperado de una variable aleatoria

Para describir completamente el comportamiento de una variable aleatoria, es necesario conocer la función de densidad de probabilidad o la función de distribución acumulativa. En algunos casos es interesante conocer algunos parámetros que resumen la información contenida en estas dos funciones.

El valor esperado o media de una variable aleatoria X, ofrece una medida de tendencia del comportamiento de la misma y se define matemáticamente de la siguiente manera [8]:

$$E[X] = \int_{\infty}^{-\infty} t f_X(t)dt \tag{2.24}$$

Si X es una variable aleatoria discreta, se define de la siguiente manera:

$$E[X] = \sum_k x_k p_x(x_k) \tag{2.25}$$

El valor esperado $E[X]$, por si mismo, aporta poca información sobre la variable aleatoria X. Por ejemplo, si sabemos que $E[X] = 0$, puede ser que X sea cero todo el tiempo o que sea igualmente probable que X tome valores positivos y negativos muy grandes. Entonces, nos interesa no sólo la media de la variable aleatoria, sino una medida de dispersión alrededor de la media.

Se define la desviación de X como

$$D = X - E[X] \tag{2.26}$$

D puede tomar valores positivos y negativos. Dado que nos interesa únicamente la magnitud de las variaciones es conveniente trabajar con D^2 que es siempre positiva.

La varianza de la variable aleatoria X está definida como la variación cuadrática media $E[D^2]$:

$$VAR[X] \equiv \sigma^2 = E[(X - E[X])^2] \tag{2.27}$$

Esta ecuación se puede simplificar de la siguiente manera:

$$\begin{aligned} VAR[X] =& E[X^2 - 2E[X]X + E[X]^2] \\ =& E[X^2] - 2E[X]E[X] + E[X]^2 \\ =& E[X^2] - E[X]^2 \end{aligned} \tag{2.28}$$

Si se calcula la raíz cuadrada de la varianza se obtiene una medida que está en las mismas unidades que X. Entonces, la desviación de una variable aleatoria X se define como [8]

$$STD[X] = VAR[X]^{1/2} = \sigma \tag{2.29}$$

La desviación estándar de una variable aleatoria se utiliza como una medida de dispersión de una distribución.

2.1.4. Segmentación

Una de las tareas primeras y mas complicadas dentro del campo de la visión, es la segmentación. La segmentación divide una imagen en diferentes regiones u objetos. El nivel hasta el cual se divide la imagen depende del problema que se va a resolver. [4]

Los algoritmos de segmentación, en general, se basan en dos propiedades básicas de los valores de intensidad de una imagen: discontinuidad y similaridad. Cuando se utiliza la discontinuidad, la idea es particionar una imagen basándose en cambios abruptos de la intensidad, que pueden estar dados por bordes. Si se utiliza la similaridad, se particiona la imagen en regiones que sean similares de acuerdo a criterios predefinidos (color, textura, intensidad, etc.). *Thresholding*, *region growing* (crecimiento de reciones) y *region splitting* son ejemplos del uso de similaridad para la segmentación. [4]

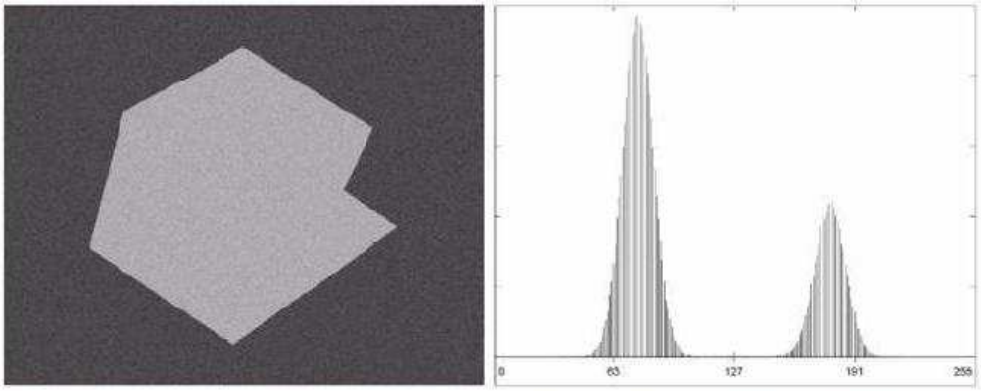

Figura 2.3: Histograma de una imagen en escala de grises, con dos regiones dominantes

Thresholding

El método de *thresholding* resulta muy intuitivo y fácil de implementar, por lo que es muy popular para aplicaciones de segmentación de imágenes.

Supongamos que se tiene un histograma en escala de grises que corresponde a una imagen, $f(x, y)$. Si la imagen está compuesta por objetos luminosos sobre un fondo obscuro, los pixeles del fondo y de los objetos estarán agrupados en el histograma en dos regiones dominantes (figura 2.3). Una manera muy obvia de separar los objetos del fondo es seleccionar un umbral T que separe estas dos regiones del histograma. Entonces, todos los puntos (x, y) para los cuales $f(x, y) > T$ corresponderán a pixeles dentro de un objeto; en otro caso, sera parte del fondo. [4]

En la figura 2.4 se muestra un caso un poco mas general, en donde se tienen tres regiones dominantes que caracterizan al histograma (por ejemplo, dos tipos de objetos con luminosidades diferentes y un fondo obscuro). En este caso, se utilizará un *thresholding* multinivel para clasificar al punto (x, y); si $T_1 < f(x, y) \leq T_2$ el pixel pertenece a un objeto, si $f(x, y) > T_2$ el pixel pertenece al otro objeto, y si $f(x, y) \leq T_1$ el pixel pertenece al fondo. En general, los problemas de segmentación que requieren umbrales múltiples, se resuelven mejor utilizando métodos de *region growing*. [4]

Una de las maneras mas utilizadas de encontrar los umbrales es analizar el histograma de niveles de grises y tratar de localizar un mínimo. Si se encuentra un mínimo

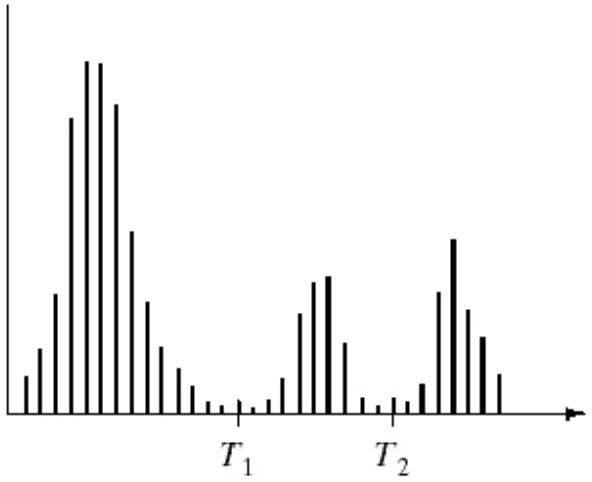

Figura 2.4: Histograma de una imagen con umbrales múltiples

significativo entre los dos picos, éste se interpreta como el umbral. En la figura 2.4, el pico de la izquierda corresponde al fondo oscuro y el pico de la derecha corresponde a los objetos claros. [2]

Crecimiento de regiones

Como el nombre lo indica, *crecimiento de regiones* es un método que agrupa los pixeles o subregiones en regiones más grandes, basándose en criterios predefinidos. Se empieza con un juego de *puntos semilla* y a partir de éstos crecen regiones al anexarse a la semilla pixeles vecinos que tengan propiedades similares a las de la semilla (por ejemplo un cierto intervalo en escala de grises o color). [4]

La elección de estas *semillas* a menudo depende de la naturaleza del problema. Cuando no existe información a priori es necesario calcular, para cada pixel, las propiedades que serán utilizadas más tarde para anexar los pixeles a una región en el proceso de crecimiento. Si el resultado de estos cálculos muestran un cúmulo de valores similares, entonces se puede utilizar como *semilla* a los pixeles que se encuentren en el centroide del cúmulo.

La elección del criterio de similaridad que se va a utilizar depende, no sólo del problema en particular, sino del tipo de imagen con el que se esté trabajando. En este trabajo en particular es necesario segmentar por color y se cuenta con una imagen

RGB y su correspondiente transformación a HSI, por lo que el criterio de similaridad dependerá principalmente de las componentes H (matiz) y S (saturación), que son las que tienen la información del color.

Otro problema en este método es plantear una condición de paro. El crecimiento de una región debe parar cuando no hay más pixeles que satisfagan el criterio de inclusión en esa región. Los criterios como nivel de gris, color o textura, no toman en cuenta la "historia" del crecimiento de la región. Hay otros criterios que hacen este procedimiento más potente, los cuales utilizan también la similaridad entre el pixel candidato y los pixeles ya incluidos en la región, y la forma de la región en crecimiento.

Planteamiento básico de los métodos de segmentación basados en regiones [4]:

Sea R toda la imagen. La segmentación se puede ver como un proceso que particiona a la región R en n subregiones, $R_1, R_2, ..., R_n$, tales que

$$\begin{aligned}
&(a) \quad \bigcup_{i=1}^{n} R_i = R. \\
&(b) \quad R_i \ \ es\, una\, region\, conectada,\ i = 1, 2, ..., n. \\
&(c) \quad R_i \cap R_j = \oslash, \quad para\, toda\, i\, y\, j,\ i \neq j. \\
&(d) \quad P(R_i) = VERDADERO, \quad para\, i = 1, 2, ..., n. \\
&(e) \quad P(R_i \cup R_j) = FALSO, \quad para\, i \neq j.
\end{aligned} \tag{2.30}$$

La condición 2.30.a indica que la segmentación debe ser completa; es decir, cada pixel de la imagen debe pertenecer a alguna región. La condición 2.30.b implica que los puntos de una región deben de estar conectados. La condición 2.30.c indica que las regiones no pueden tener intersección. La condición 2.30.d se refiere a las propiedades que deben satisfacer los pixeles en una región segmentada (por ejemplo $P(R_i)$ = VERDADERO si todos los pixeles en R_i tienen el mismo nivel de gris. Por último, la condición 2.30.e indica que las regiones R_i y R_j son diferentes en tanto que los pixeles de cada región tienen diferentes propiedades, de acuerdo al criterio de pertenencia para cada región.

Conectividad

La conectividad entre pixeles es un concepto fundamental que simplifica la definición de numerosos conceptos en imágenes digitales, tales como regiones y fronteras. Para establecer que dos pixeles están conectados, se debe determinar si son vecinos

y que algunas de sus propiedades cumplan con un criterio de similaridad.

Un pixel p con coordenadas (x, y) tiene cuatro vecinos horizontales y verticales cuyas coordenadas están dadas por [4]

$$(x+1, y), (x-1, y), (x, y+1), (x, y-1) \tag{2.31}$$

Este juego de cuatro pixeles es llamado 4-vecindad de p, y se denota por $N_4(p)$. Cada pixel es una unidad de distancia y algunos de los vecinos de p pueden encontrarse afuera de la imagen digital si (x, y) está en el borde de la imagen.

Los cuatro vecinos diagonales de p tienen las coordenadas

$$(x+1, y+1), (x+1, y-1), (x-1, y+1), (x-1, y-1) \tag{2.32}$$

y se denotan como $N_D(p)$. Estos puntos junto con los 4-vecinos, son llamados 8-vecindad de p, denotado como $N_8(P)$.

Supongamos que se esta trabajando una imagen en escala de grises. Sea V el juego de valores en escala de gris usado para definir la adyacencia. En una imagen monocromática V sería un conjunto con un solo elemento, ya sea $V = \{0\}$ o $V = \{1\}$. En una imagen en escala de grises, V es generalmente un conjunto de elementos que pueden expresarse como un intervalo de valores en la escala de grises ($V = I \in [0, 255] : I_0 < I < I_1$). A continuación se describen dos tipos de conectividad [4]:

- (a) 4-conexidad. Dos pixeles p y q que tengan valores contenidos en V son 4-conexos si q esta en la vecindad $N_4(p)$. (Ver figura 2.5 izquierda)
- (b) 8-conexidad. Dos pixeles p y q que tengan valores contenidos en V son 8-conexos si q esta en la vecindad $N_8(p)$. (Ver figura 2.5 derecha)

2.1.5. *Run-length Encoding* (RLE)

Run-length encoding (RLE) es un algoritmo de compresión de datos sin pérdidas, en la que secuencias de datos contiguos con el mismo valor se reemplazan por el dato y la cantidad de veces que estaba repetido. Esta manera de almacenar, resulta particularmente útil cuando se trabaja con datos que contienen muchas secuencias de datos iguales. A estas secuencias de datos iguales se les llama "corridas" (*run*). A

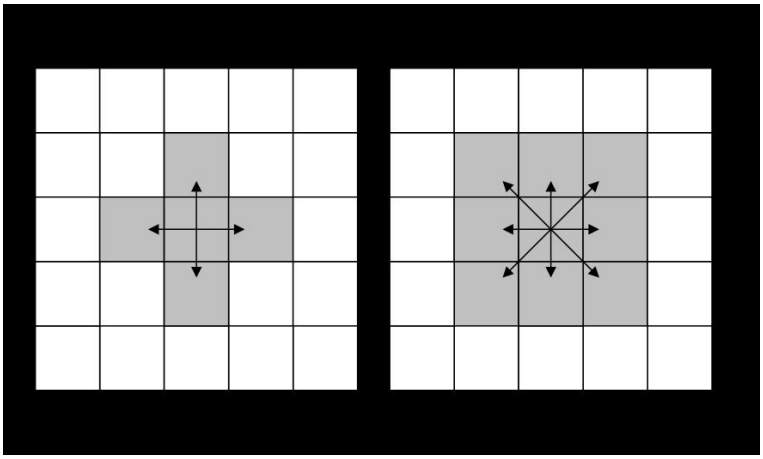

Figura 2.5: Izquierda: 4-conexidad. Derecha: 8-conexidad.

continuación se muestra un ejemplo. [26]

Si se le aplica el algoritmo RLE a la secuencia de datos:

RRRVVVVVVVVVAAAVVVVVVRRR

la secuencia comprimida sería la siguiente:

3R8V3A6V3R

Esto se puede interpretar como tres ROJOS, ocho VERDES, tres AZULES, seis VERDES y tres ROJOS.

2.1.6. *BLOB (Binary Large Object)*

Según la definición formal, un BOLB es una colección de datos binarios almacenados en una sola entidad en un sistema de manejo de bases de datos. Los *BLOBs* son típicamente imágenes, audio u otros objetos multimedia. [23]

Los *BLOBs* originalmente eran pedazos de datos almacenados en una base de datos inventados por Jim Starkey. Posteriormente Terry McKiever sintió la necesidad de hacer un acrónimo inventado como Basic Large Object. Después Informix inventó un acrónimo alternativo, Binary Large Object.

El tipo de datos y la definición fueron introducidos para describir datos no definidos en los sistemas computacionales de bases de datos tradicionales.[23]

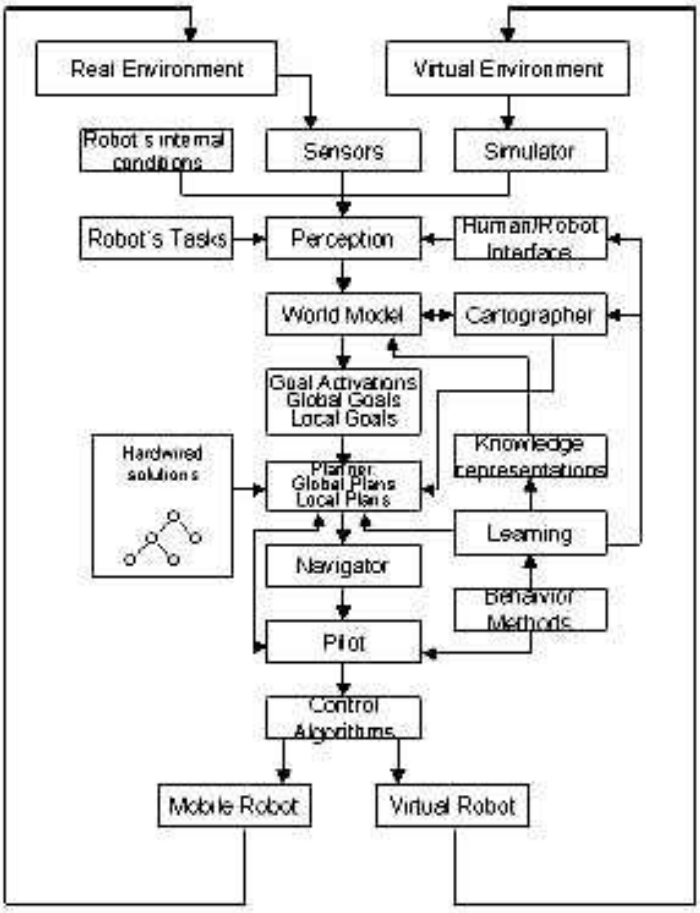

Figura 2.6: Arquitectura del Sistema *ViRbot*

2.2. Arquitecturas para la inteligencia del robot

2.2.1. Arquitectura *ViRbot*

Hay varias arquitecturas que se pueden utilizar para controlar un robot móvil. En este trabajo se utiliza una arquitectura similar a la del sistema *ViRbot* [15, 17], en donde la operación del robot se descompone en varias capas (figura 2.6), cada una de las cuales desempeña una función específica en el control del comportamiento global del robot.

Para este trabajo resultan particularmente importantes dos de las capas de esta arquitectura: el planeador y el navegador. A continuación se describen brevemente.

Planeador

El problema básico de la planeación de los movimientos de un robot se reduce a que, dada una posición y orientación inicial, se debe encontrar una serie de posiciones y orientaciones para el robot (de manera que se eviten colisiones con los posibles obstáculos del entorno) comenzando en la posición y orientación inicial y terminando en la posición y orientación deseada. Si no se puede encontrar dicha trayectoria, se debe reportar que resulta imposible resolver el problema particular. [17]

La planeación se define como la formulación de un procedimiento o una guía para alcanzar algún objetivo o desarrollar alguna tarea. Esto requiere que se busquen configuraciones para encontrar una trayectoria que corresponda a una serie de operaciones que resolverán el problema.

El planeador recibe como entrada el estado inicial y final, además de una descripción del entorno y genera un plan global que llevará al robot desde el estado inicial hasta el estado final. Por ejemplo, en el caso en que el comando sea mover el robot de un cuarto a otro, el planeador encontrará el mejor conjunto de movimientos entre cuartos que deba efectuar el robot para que llegue al destino final. Dentro de cada cuarto deberá encontrar también la mejor trayectoria de movimientos, considerando los obstáculos conocidos. Los obstáculos que encuentra el robot representan los objetos en los cuartos del entorno real. [17]

En resumen, el planeador en un robot móvil se encarga de trazar una trayectoria para que el robot se desplace desde una posición/orientación inicial a una posición/orientación final, evadiendo los obstáculos conocidos que puedan presentarse en su camino.

Navegador

El problema de la navegación para un robot móvil es, en primera instancia, trazar una ruta de la posición actual a la posición de destino evitando los obstáculos conocidos (planeador), y luego mover el robot al objetivo teniendo el cuidado de no colisionar con los obstáculos no conocidos que se puedan encontrar en el camino (navegador). [6]

El navegador deberá encargarse de ejecutar los movimientos encontrados por el planeador para seguir la ruta trazada, pero deberá monitorear constantemente los sensores para evitar colisiones con obstáculos no conocidos por el planeador.

2.2.2. Paradigmas de la robótica: *jerárquico, reactivo e híbrido*

Actualmente, existen tres principales paradigmas para estructurar la inteligencia de los robots: *jerárquico*, *reactivo* e *híbrido (deliberativo/reactivo)*. Existen tres primitivas comúnmente aceptadas en la robótica, a partir de las cuales se estudian estos tres paradigmas: *sentir*, *planear* y *actuar*. [10]

El paradigma *jerárquico* o *deliberativo* es el más viejo de los paradigmas y se utilizó desde 1967 hasta 1990. Este paradigma está basado en una visión introspectiva de como piensa el ser humano: se realiza una planeación para cada situación en la que se encuentre el robot. El robot detecta el mundo real con sus sensores, luego planea su siguiente acción y finalmente actúa (*sentir* $\rightarrow$ *planear* $\rightarrow$ *actuar*); en cada paso se planea la siguiente acción que se llevará a cabo. Una característica importante de este paradigma es que toda la información obtenida con los sensores se integra en un modelo global del mundo, que es una representación que utiliza el planeador.

Gracias a la psicología cognitiva ahora se sabe que la introspección no siempre es la mejor manera de resolver un problema. En vez de esto, se tienen comportamientos predefinidos para resolver ciertos problemas cuando se detecta alguna situación en particular. En ocasiones, la información detectada por los sensores puede estar directamente ligada con una acción.

El paradigma *reactivo* fue una respuesta en contra del paradigma *jerárquico* y llevó a grandes avances en la robótica. Se utilizó mucho desde 1988 hasta 1992 y actualmente todavía se utiliza, pero hay una fuerte tendencia hacia el modelo *híbrido*. El paradigma *reactivo* surge del interés por investigar en el campo de biología y psicología cognitiva.

El paradigma *reactivo* descarta por completo la planeación, solo siente y actúa, mientras que en el paradigma *jerárquico* el actuar siempre es el resultado de un plan. En el modelo *reactivo* actuar es el resultado directo de haber detectado (*sentido*) algo.

El robot tiene varias parejas de *sensaciones-actos* ligados. Éstas se ejecutan en

procesos concurrentes llamados comportamientos, que recogen la información y calculan cual es la mejor acción independientemente de que otro proceso se esté ejecutando. Dos comportamientos pueden activarse y cada uno dirigir al robot hacia direcciones diferentes. Por ejemplo, un comportamiento dicta que el robot debe moverse hacia el frente para llegar a un objetivo, mientras que otro comportamiento dice que se debe girar 90° para evitar chocar con un obstáculo en el camino. El robot ejecutará una combinación de ambos comportamientos, tomando una dirección de 45° en la que evita la colisión, pero sigue acercándose al objetivo. Nótese que ninguno de los dos comportamientos decidió que se girara 45°; este comportamiento emergió de la combinación de otros dos comportamientos.

El paradigma *reactivo* resultó muy útil para imitar comportamientos sencillos, por ejemplo de algunos animales, pero no fue lo suficientemente eficaz para explicar la inteligencia humana. A pesar de ésto, tiene propiedades muy deseables, como lo es la rápida ejecución de los comportamientos al haberse eliminado la planeación.

El paradigma *híbrido (deliberativo/reactivo)* surge en los noventas y sigue siendo el área de investigación actual. Con este paradigma, el robot primero planea (delibera) como descomponer la tarea en subtareas (planeación de misión) y luego calcula cuáles son los mejores comportamientos para desempeñar esta subtarea. En ese punto, los comportamientos se ejecutan como si se estuviera en un modelo *reactivo*. En este paradigma la organización es $planear \rightarrow sentir-actuar$; la planeación se realiza en un primer paso y luego sentir y actuar se llevan acabo juntos. Los datos de los sensores se tienen que hacer llegar a cada comportamiento que necesite de ese sensor, pero también están disponibles para el planeador, en donde se construye un modelo global del mundo orientado a tareas. El planeador puede monitorear los datos sensados por cada comportamiento para agregar información al mapa del mundo. [10]

2.3. Campos potenciales

Los comportamientos basados en *campos potenciales* utilizan vectores para representar los comportamientos y la suma de vectores para combinar las conductas y generar otro comportamiento emergente.

Un campo potencial es un arreglo o campo de vectores. Un vector es una estructura matemática que consta de magnitud y dirección. Éstos pueden ser utilizados para representar algún tipo de fuerza y son dibujados típicamente como una flecha, cuya longitud indica la magnitud de la fuerza y el ángulo, la dirección. [10]

En la planeación de trayectorias utilizando *campos potenciales* el robot se considera como una partícula que se mueve inmersa en un campo potencial generado por el objetivo y por los obstáculos presentes en el entorno. El objetivo genera un potencial atractivo y los obstáculos generan un potencial repulsivo. Un campo potencial puede ser visto como un campo de energía, por lo que su gradiente en cada punto es una fuerza. El robot inmerso en este campo potencial esta sometido a la acción de una fuerza que lo acerca a su objetivo mientras se mantiene alejado de los obstáculos. [12]

El movimiento, en los métodos basados en *campos potenciales*, se puede interpretar como el movimiento de una partícula en un campo vectorial del gradiente del potencial generado por partículas eléctricas positivas y negativas. En esta analogía, el robot es una carga positiva, el objetivo es una carga negativa y los obstáculos son cargas positivas. Los gradientes, entonces, se pueden interpretar como fuerzas que atraen a la partícula (robot) con carga positiva hacia una partícula negativa que representa el objetivo. Los obstáculos actúan como cargas positivas que generan fuerzas repulsivas que obligan a la partícula (robot) a alejarse de ellos. La combinación de la fuerza de atracción hacia el objetivo y de la fuerza de repulsión de los obstáculos, llevarán al robot hacia el objetivo siguiendo una trayectoria segura.

Los *campos potenciales* también pueden verse como un terreno con elevaciones generadas por los obstáculos y valles con una pendiente, cuyo punto más bajo representa el objetivo del robot. El robot sigue la trayectoria a través del negativo del gradiente de la función potencial; ésto quiere decir que descenderá hacia el punto más bajo del valle. Con esta analogía se ve claro que el robot puede llegar a quedar atrapado en un mínimo local alejado del objetivo. Ésta es una de las mayores deficiencias de los métodos de *campos potenciales*.

A continuación se desarrollan las ecuaciones que gobiernan este modelo:

Sea q la posición del robot, considerado como una partícula moviéndose en el plano R^2. La posición se define como la dupla $q = (x, y)$. El campo potencial artificial en donde el robot se mueve es una función escalar $U(q)$: $R^2 \rightarrow R$ generado por la superposición de potenciales atractivos y repulsivos

$$U(q) = U_{at}(q) + U_{rep}(q). \tag{2.33}$$

El potencial repulsivo es el resultado de la superposición de los potenciales repulsivos de cada obstáculo. La ecuación 2.33 se puede reescribir de la siguiente manera:

$$U(q) = U_{at}(q) + \sum_i U_{rep_i}(q), \tag{2.34}$$

donde $U_{rep_i}(q)$ representa el potencial repulsivo generado por el i-ésimo obstáculo.

Considérese que $U(q)$ es derivable. Para cada q, el gradiente del campo potencial, denotado por $\nabla U(q)$, es un vector que apunta en la dirección hacia donde $U(q)$ tiene su máxima variación (local). En los métodos de navegación basados en *campos potenciales*, el potencial atractivo se define como cero en el objetivo y se incrementa conforme el robot se aleja del mismo. El potentical repulsivo asociado a cada obstáculo es muy grande en las cercanías de los obstáculos y decrece cuando la distancia a los mismos aumenta. Estos principios son utilizados como restricciones para elegir la función potencial atractiva y repulsiva respectivamente.

La fuerza que mueve al robot es el negativo del gradiente del campo potencial:

$$F(q) = F_{at}(q) + F_{rep}(q) = -\nabla U_{at} - \nabla U_{rep}(q). \tag{2.35}$$

La fuerza $F(q)$ en la ecuación 2.35 es un vector que apunta en la dirección que, para cada q, el campo potencial decrece más. Esta fuerza puede ser considerada como el vector de velocidad que mueve al robot puntual. [12]

Potencial atractivo

La elección típica para el potencial atractivo es una parábola que crece cuadráticamente con el cuadrado de la distancia al objetivo [12],

$$U_{at}(q) = \frac{1}{2} k_{at} d_{obj}^2(q) \tag{2.36}$$

donde $d_{obj}(q) = \|q - q_{obj}\|$ es la distancia euclidiana del robot al objetivo (q_{at}) y k_{at} es un factor de escala que se utiliza para calibrar la magnitud de la fuerza de atracción. El gradiente

$$\nabla U_{at}(q) = k_{at}(q - q_{obj}) \tag{2.37}$$

es un campo vectorial proporcional al vector $q - q_{obj}$ que apunta en dirección opuesta a q_{obj}. Mientras mas lejos se encuentre el robot del objetivo será mas grande la magnitud del campo potencial atractor. La fuerza atractora será el negativo del gradiente del potencial, de manera que el vector apunte en dirección al objetivo

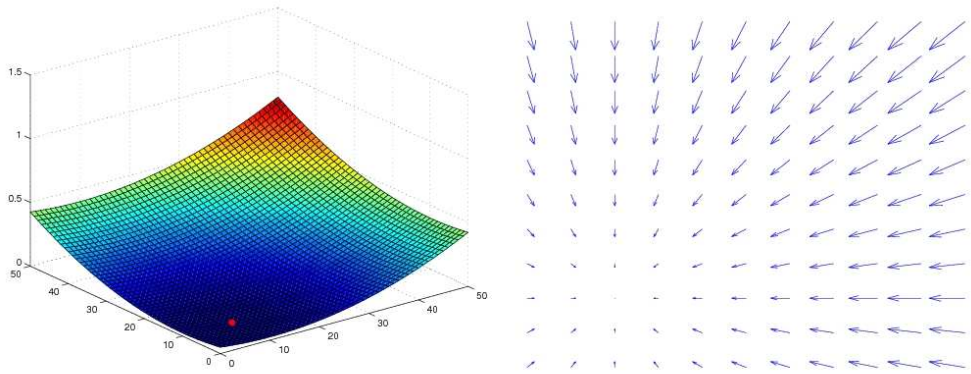

Figura 2.7: Izquierda: Potencial atractor. Derecha: Fuerza atractora hacia el objetivo

$$F_{at}(q) = -\nabla U_{at}(q) = -k_{at}(q - q_{obj}). \tag{2.38}$$

Al hacer la velocidad del robot proporcional al campo de fuerza, la fuerza dada por la ecuación 2.38 lleva al robot hacia el objetivo con una velocidad que decrece mientras éste se acerca al mismo. En esta ecuación la fuerza depende linealmente de la distancia al objetivo, lo cual quiere decir que dicha fuerza aumenta sin límites conforme el robot se aleja. Cuando el robot se encuentra muy lejano a su objetivo, la fuerza impone que el robot se acerque a él rápidamente. Por el contrario, la fuerza (al igual que la velocidad) tiende a cero cuando el robot se aproxima al objetivo. Esto evita que el robot llegue al objetivo con una velocidad muy alta y tenga dificultad para frenar [12].

La figura 2.7 representa el potencial atractivo y el campo vectorial de fuerza (negativo del gradiente del potencial) para una situación en donde se tiene un solo atractor en una posición dada (marcada en la figura con un punto) y ningún repulsor.

Potencial repulsor

El potencial repulsor mantiene al robot alejado de los obstáculos. Éste se hace más fuerte cuando el robot se acerca al obstáculo y decrece cuando se encuentra lejos. Por el principio de superposición, el potencial repulsivo total es la suma de los potenciales repulsivos generados por todos los obstáculos [12],

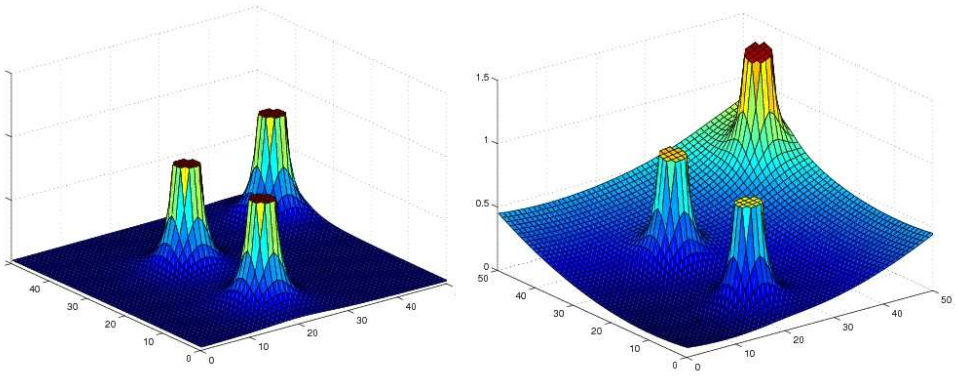

Figura 2.8: Izquierda: Potencial repulsor. Derecha: Suma de potencial atractor y repulsor.

$$U_{rep} = \sum_i U_{rep_i}(q) \tag{2.39}$$

Es razonable considerar que la influencia de un obstáculo está limitada a un espacio pequeño en su vecindad, hasta una cierta distancia. Un obstáculo muy lejano al robot no debe repelerlo. Por otro lado, la magnitud del potencial repulsor deberá aumentar cuando el robot se aproxime al obstáculo. Tomando en cuenta estas dos restricciones, una posible elección para el potencial repulsivo generado por un obstáculo es

$$U_{rep_i}(q) = \begin{cases} \frac{1}{2} k_{obst_i} \left(\frac{1}{d_{obst_i}(q)} - \frac{1}{d_0} \right)^2 & si\ d_{obst_i}(q) < d_0 \\ 0 & si\ d_{obst_i}(q) \geq d_0, \end{cases} \tag{2.40}$$

donde $d_{obst_i}(q)$ es la distancia mínima entre q y el i-ésimo obstáculo; k_{obst_i} es un factor de escala que se utiliza para calibración y d_0 es la distancia umbral de influencia del obstáculo.

El negativo del gradiente del potencial repulsivo $F_{rep_i}(q) = -\nabla U_{rep_i}(q)$ está dado por [12]:

$$F_{rep_i}(q) = \begin{cases} k_{obst_i} \left(\frac{1}{d_{obst_i}(q)} - \frac{1}{d_0} \right) \frac{1}{d^2_{obst_i}(q)} \frac{q - q_{obst_i}}{d_{obst_i}} & si\ d_{obst_i}(q) < d_0 \\ 0 & si\ d_{obst_i}(q) \geq d_0. \end{cases} \tag{2.41}$$

En la figura 2.8(izquierda) se muestra el potencial repulsor generado por tres obstáculos. La figura 2.8(derecha) muestra el potencial final, que es la suma de los potenciales repulsores y el potencial atractor. En estas figuras se observa claramente que si el robot fuera una canica libre en algún punto de la superficie, ésta rodaría en dirección del objetivo, que es el punto de menor potencial. El camino que seguiría, sería a través de la ruta con la pendiente negativa más grande. [12]

2.4. Estructuras de datos: Listas ligadas

En ciencias de la computación, una lista ligada es una de las estructuras de datos fundamentales utilizada para la programación. Una lista es una secuencia de estructuras llamadas nodos que contienen por lo menos dos campos: uno arbitrario, con un dato que se quiera guardar, y otro que contiene un apuntador (liga o referencia) que apunta al siguiente nodo de la lista. Una lista ligada es un tipo de dato *auto-referencial*, porque contiene un apuntador que apunta a otro dato del mismo tipo. [24]

Las listas ligadas permiten la inserción y remoción de datos en cualquier punto de la lista; además, a diferencia de los arreglos, no es necesario definir un tamaño máximo con el que se va a trabajar, puesto que cada que se inserta un nodo, se asigna memoria para el mismo y luego se incorpora a la lista. Las listas ligadas tienen como desventaja que no se puede hacer acceso aleatorio; solo se puede hacer acceso secuencial; es decir, para acceder al vigésimo elemento de la lista será necesario acceder primero a los 19 elementos anteriores. [24]

En la figura 2.9 se muestra un esquema de una lista ligada en donde se observan los nodos con sus dos campos y las ligas (apuntadores) entre ellos.

Dado que la lista ligada es un tipo de dato abstracto, debe cumplir con el concepto de encapsulamiento, de tal manera que una lista ligada debe contar también con métodos o funciones de acceso que definen las operaciones que se pueden realizar con dicho tipo de dato. Los campos internos de las estructuras nodo, no deberán ser accesados sino a través de los métodos o funciones de acceso que se implementen.

Los métodos principales que se implementan comúnmente para una lista ligada son los siguientes [7]:

- Crear una nueva lista vacía

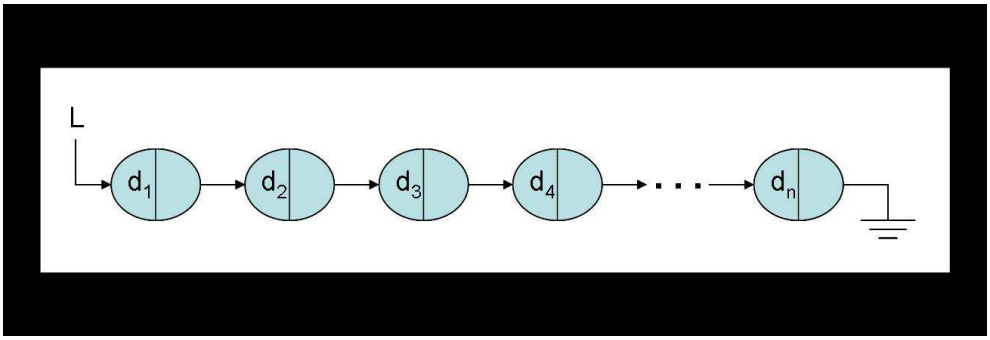

Figura 2.9: Diagrama de una lista ligada

- Insertar un dato
- Eliminar un dato
- Consultar el primer elemento de la lista
- Avanzar al nodo siguiente
- Consultar si la lista esta vacía

2.5. Comunicaciones con *sockets*

Los *sockets* son una de las tecnologías fundamentales en las redes de computadoras. Éstos permiten que las aplicaciones se comuniquen utilizando mecanismos estandarizados construidos en el hardware de red y en los sistemas operativos.

Los *sockets* son una interfaz simple que permite escribir aplicaciones de red con facilidad. Un *socket* es similar a un *descriptor de archivo*, ya que identifica a un canal de comunicación como un entero positivo. Es un manejador asociado con un conjunto de datos guardado en en la implementación del protocolo de red. Los datos asociados con el *socket* son, por ejemplo, la dirección IP, los puertos para ambos lados de la conexión y el protocolo de transporte de la conexión (i.e. TCP o UDP).[13]

Un *socket* representa una conexión entre exactamente dos piezas de software. Los programas que utilizan *sockets*, en general, corren en dos computadoras diferentes de la red, pero los *sockets* pueden también ser utilizados para comunicarse localmente entre procesos en una misma computadora. Los *sockets* son bidireccionales, lo cual

quiere decir que en ambos lados de la conexión se pueden enviar y recibir datos. En ocasiones, se le llama *cliente* a la aplicación que inicia la comunicación y *servidor* a la otra.

Las interfaces de los *sockets* se pueden dividir en dos categorías principales: la más utilizada es la de los *stream sockets* (*sockets* TCP), que implementan una comunicación orientada a conexión. Esencialmente, un *stream* (flujo de datos) necesita que las dos entidades que se van a comunicar establezcan primero una comunicación, después de lo cual se garantiza que los datos que se envíen a través de la conexión sean recibidos en el mismo orden en el que se mandan.

Otra categoría son los *datagramas* (*sockets* UDP), que ofrecen una comunicación no orientada a conexión. Con los datagramas las conexiones son implícitas. En cada lado de la conexión se envían datagramas cuando se necesita y se espera a que el otro lado responda; los mensajes pueden perderse en el trayecto o recibirse en desorden, pero es la responsabilidad de la aplicación manejar estos problemas. Los *sockets* que usan datagramas pueden tener algunas ventajas sobre los *stream sockets*, por ejemplo, en aplicaciones de transmisión de video, en donde resulta muy importante que la aplicación no se detenga en ningún momento. Si un paquete se pierde o llega en desorden es descartado y se siguen procesando los demás paquetes, en vez de esperar a que sea reenviado.

Los sistemas operativos modernos, generalmente, proveen una implementación de alguna librería y APIs (*application programing interfaces*) para el uso de *sockets*. Dos de las APIs más utilizadas son los Sockets Berkeley para sistemas UNIX, y otros son los *sockets* Winsock, que es una librería para los sistemas operativos Windows. [13]

2.5.1. Implementación de una aplicación con *sockets* TCP

En la figura 2.10 se muestra el diagrama de flujo para una aplicación cliente-servidor TCP.

Ambos, cliente y servidor, deben crean un *socket* mediante la función *socket()*, para poder comunicarse. [13]

El servidor llama a *bind()* para darle un nombre al *socket*, para luego poder recibir conexiones; es decir, se establece el número de puerto que la aplicación monitoreará para recibir conexiones.

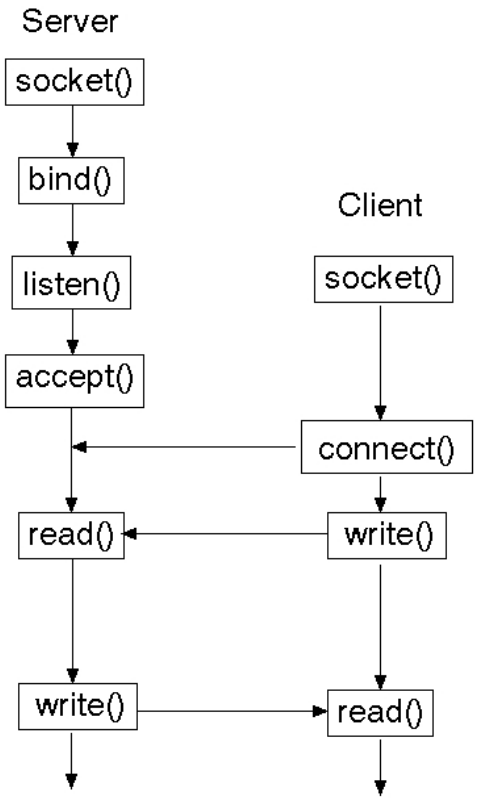

Figura 2.10: Diagrama de flujo de una aplicación cliente-servidor con *sockets* TCP

El servidor habilita su *socket* para poder recibir conexiones, llamando a la función *listen()*. El cliente no necesita realizar este paso porque no va a recibir conexiones, solo intentará conectarse con el servidor.

El servidor ejecuta la función *accept()* y queda en estado de espera, la función *accept()* no retorna hasta que intenten conectarse.

El cliente usa la función *connect()* para realizar el intento de conexión, en ese momento, la función *accept()* del servidor retorna con un parámetro que es un nuevo descriptor de *socket*, el cual se utiliza para realizar la transferencia de datos por la red con el cliente.

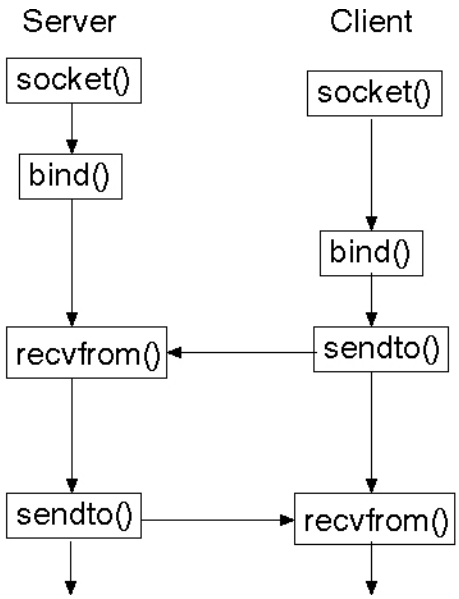

Figura 2.11: Diagrama de flujo de una aplicación que utiliza *sockets* UDP

Una vez establecida la conexión se utilizan las funciones *write()* y *read()* (o *send()* y *recv()*) con el descriptor de *socket* del paso anterior para realizar la transferencia de datos.

Para finalizar la conexión se utilizan las funciones *close()* o *shutdown()*. [13]

2.5.2. Implementación de una aplicación con *sockets* UDP

En la figura 2.11 se muestra el diagrama de flujo para una aplicación que utiliza *sockets* UDP.

Ambos, cliente y servidor, crean un *socket* mediante la función *socket()*. [13]

El servidor debe establecer por qué número de puerto recibirá los datos utilizando la función *bind()*; en este caso no existe la conexión, los datos se envían como si fueran mensajes.

Para realizar transferencia de datos se utilizan las funciones *sendto()* y *recvfrom()*.

Para finalizar la conexión se utilizan las funciones *close()* o *shutdown()*. [13]

Capítulo 3

Desarrollo del sistema de visión

El sistema de visión que se describe en el presente capítulo fue desarrollado en conjunto con otro proyecto de tesis [14] desarrollado también en el laboratorio de Biorobótica del Posgrado de la Facultad de Ingeniería.

El sistema de visión se dividió en ocho módulos:

- Captura de video con DirectX: se encarga de realizar la captura del video utilizando librerías de DirectX
- Conversión de espacio de color RGB a HSI: se encarga de cambiar el mapa de color de RGB a HSI
- Calibración: establece los umbrales para la segmentación
- Segmentación: clasifica los pixeles según su color
- *Run Length Encoding* (RLE): agrupa pixeles adyacentes del mismo color
- Formación de *BLOBs*: agrupa en regiones bidimensionales a los pixeles agrupados con RLE
- Detección de robots y orientación: a partir de los *BLOBs* detecta a los diferentes robots y calcula su orientación
- Comunicaciones: envía utilizando *sockets* UDP las posiciones de los robots y la pelota al sistema de planeación/navegación

Este sistema se programó en el lenguaje C usando el ambiente de desarrollo Visual Studio 6, junto con las librerías de captura de video de DirectX.

3.1. Conversión de espacio de color RGB a HSI

Como se discutió en el marco teórico, el espacio de color HSI resulta más indicado que el RGB para poder realizar la segmentación por umbrales. Es por eso que fue necesario desarrollar un módulo que se encargara de realizar dicha conversión.

El algoritmo para la conversión es el siguiente:

Sean R, G y B las tres componentes de color en el espacio RGB y H, S e I las componentes en el espacio HSI. Sea m el mínimo valor entre R, G y B.

$$S = 255 - m\left(\frac{765}{R+G+B}\right) \tag{3.1}$$

$$I = \frac{R+G+B}{3} \tag{3.2}$$

$$H = cos^{-1}\left(\frac{2R-G-B}{(R-G)^2+(R-B)(G-B)}\right) \tag{3.3}$$

3.2. Calibración

El módulo de calibración del sistema de visión tiene como objetivo fijar los umbrales que se utilizarán en el módulo de segmentación para hacer la clasificación de los pixeles por colores.

El usuario deberá seleccionar el color que se va a calibrar y, a continuación, buscará en el video capturado por la cámara las regiones de la imagen cuyo color sea el que está en proceso de calibración. A continuación deberá dar click en tres regiones (de preferencia distantes) con este color, de manera que se contemplen las diferencias en iluminación que habrá en las diferentes zonas de la imagen.

Para los tres puntos de la imagen donde el usuario hizo click con el ratón, se consideran las regiones formadas por un cuadrado de cinco por cinco pixeles alrededor del punto señalado; se utiliza la información de todos los pixeles en estas tres regiones (75 pixeles en total) para generar el histograma para cada canal. El histograma se utiliza como función de densidad de probabilidad (*fdp*) de que un pixel tenga cierto valor en cada uno de sus canales (HSI).

Una vez obtenida la *fdp* para cada color, se obtiene el valor esperado de X, ($E[X]$), que es el primer momento de la variable aleatoria. Sabemos que $E[X]$ es un

promedio ponderado de la variable aleatoria X, por lo que este valor representa la media de la distribución.

También se calcula la varianza de la distribución, que está definida como $VAR[X] \equiv \sigma^2 = E[(X - E[X])^2]$, y la desviación estándar que se define como $\sqrt{VAR[X]} = \sigma$.

En el módulo de segmentación se utilizará la media como valor central y un intervalo de más menos dos veces la desviación estándar para realizar las comparaciones con los valores de los tres canales de cada pixel y poder encontrar su color; suponiendo que los valores de cada canal para determinado color siguen una distribución normal, el 95.4 % de los pixeles del color en cuestión se encontrarán dentro de este intervalo.

3.3. Segmentación

En el proceso de segmentación se comparan los componentes HSI de cada pixel de la imagen fija con los intervalos obtenidos en la etapa de calibración convertidos también en HSI. Si las tres componentes del pixel se encuentran dentro del intervalo correspondiente a alguno de los colores a detectar, se considera que el punto de la imagen es del color en cuestión. El resultado de este proceso es que se clasifica a cada pixel dentro de una de las siguientes categorías:

- Rojo
- Azul
- Amarillo
- Verde
- Rosa
- Cian
- Fondo

La categoría *fondo* quiere decir que el pixel no es de ninguno de los colores que nos interesa detectar, por lo que representa parte del fondo de la imagen o ruido. Estos pixeles no son de interés para el proceso de formación de regiones.

3.4. Formación de regiones (crecimiento de regiones)

Para implementar el algoritmo de regiones en crecimiento se utilizaron como *puntos semilla* a todos los pixeles que resultaron ser de alguno de los seis colores de interés (rojo, azul, amarillo, verde, rosa o cian); los pixeles del fondo son descartados.

A partir de estas *semillas* las regiones comienzan a crecer. El algoritmo de regiones en crecimiento fue dividido en dos etapas para economizar tiempo de procesamiento: la primera obtiene regiones lineales (sobre un mismo renglón) utilizando el algoritmo *Run Length Encoding (RLE)* y la segunda etapa obtiene las regiones bidimensionales (que en este trabajo llamamos *BLOBs*) uniendo las regiones formadas con RLE y utilizando el concepto de *8-conexidad*. A continuación se describen estas dos etapas.

3.4.1. Algoritmo *Run Length Encoding* (RLE)

Una vez detectado el color de cada pixel, es necesario agruparlos en una estructura para formar las diferentes regiones de color. Para empezar, se utilizó el algoritmo *Run Length Encoding* (RLE), con el que se recorre cada renglón de la imagen buscando pixeles del mismo color que estén adyacentes; éstos se van agrupando (siempre y cuando haya por lo menos dos pixeles adyacentes del mismo color) dentro de una estructura que guarda los datos de las regiones unidimensionales que se van formando:

1. Color de la región
2. Coordenada x del primer pixel de la región
3. Coordenada x del último pixel de la región
4. Coordenada y de la región

Para implementar este algoritmo se desarrolló una máquina de estados (figura 3.1), para cada uno de los colores que se quieren detectar, que recibe como entrada el color de los pixeles que se leen y agrupa en estructuras unidimensionales a los pixeles del color que se quiere detectar.

La máquina de estados consta de tres estados:

1. Primer estado:

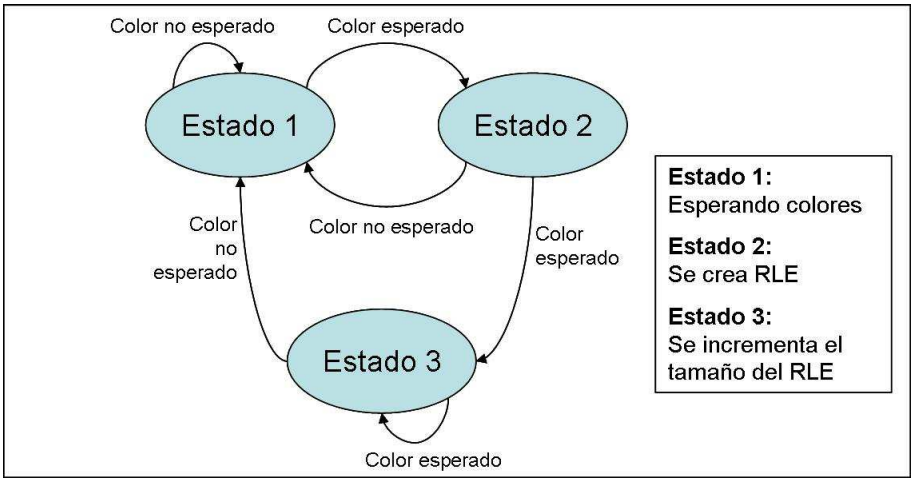

Figura 3.1: Máquina de estados para formar RLE's

- Si recibe un color que no es el esperado, la máquina permanece en el primer estado.
- Si recibe el color esperado, la máquina avanza al segundo estado.

2. Segundo estado: Se crea un nodo con las coordenadas del RLE que se está formando y se inserta en la lista.

   - Si recibe un color que no es el esperado, la máquina regresa al primer estado.
   - Si recibe el color esperado, la máquina avanza al tercer estado.

3. Tercer estado: Se incrementa en uno el tamaño de la estructura y se actualizan las coordenadas de la región.

   - Si recibe un color que no es el esperado, la máquina regresa al primer estado.
   - Si recibe el color esperado, la máquina permanece en el tercer estado.

En el sistema se utilizan siete máquinas de estados como la anterior, una para cada color que se quiere detectar.

3.4.2. Listas ligadas para RLE

Se diseñó una lista ligada en cuyos nodos se pueden almacenar las estructuras generadas con el algoritmo RLE. Se creó una estructura para almacenar RLE's cuyos campos son los siguientes:

- Coordenada x del pixel de la extrema izquierda del intervalo
- Coordenada x del pixel de la extrema derecha del intervalo
- Coordenada y del renglón del RLE en cuestión
- Color del RLE
- Número de pixeles o tamaño del RLE
- Identificador o etiqueta del RLE
- Identificador o etiqueta del *BLOB* al que pertenece este RLE

El contenido de los nodos de la lista diseñada es el siguiente:

- Apuntador al siguiente nodo de la lista
- Dato de tipo RLE

Se programaron funciones de acceso a la lista ligada, de manera que el manejo interno de los apuntadores esté encapsulado. Las funciones de acceso con las que cuenta la lista ligada diseñada son las siguientes:

- Inicializar una lista vacía
- Insertar un elemento al principio de la lista
- Crear un nuevo nodo
- Eliminar la lista ligada y liberar la memoria
- Eliminar el primer elemento de la lista
- Consultar el primer elemento de la lista

- Avanzar la lista al siguiente elemento
- Consultar si la lista está vacía
- Duplicar la lista ligada

Se crearon siete de estas listas ligadas para almacenar las estructuras RLE de cada color generadas al recorrer toda la imagen para luego ser utilizadas en la formación de los *BLOBs*.

3.4.3. Listas ligadas para *BLOBs*

Se diseñó también una lista ligada para almacenar las estructuras bidimensionales (*BLOBs*) que se forman a partir de la unión de estructuras de RLE del mismo color.

Se creó una estructura BLOB con los siguientes campos:

- Área del BLOB
- Color del BLOB
- Coordenadas del punto superior izquierdo de la región
- Coordenadas del punto inferior derecho de la región
- Centroide de la región
- Identificador o etiqueta del BLOB

Se diseñó también una estructura para almacenar los nodos de la lista:

- Apuntador al siguiente nodo de la lista
- Dato de tipo BLOB

Para encapsular el manejo de la lista ligada, ésta ofrece las siguientes funciones de acceso:

- Inicializa una lista vicia
- Inserta un elemento al principio de la lista

- Inserta un elemento en orden
- Crear un nodo nuevo
- Elimina un elemento de la lista
- Consulta el primer elemento de la lista
- Avanza la lista al siguiente elemento
- Elimina toda la lista ligada y libera la memoria
- Consulta si la lista esta vacía
- Busca un elemento con una etiqueta dada
- Duplica la lista
- Concatena dos listas ligadas

3.4.4. Formación de *BLOBs*

Ya que se cuenta con las listas ligadas que contienen las estructuras formadas con RLE, es necesario generar nuevas estructuras bidimensionales. Cada una de éstas estará formada por varias de las estructuras unidimensionales (RLE's) almacenadas en las listas.

Los *BLOBs* son regiones rectangulares en crecimiento que se obtienen a partir de la fusión de regiones de pixeles de un mismo color. La fusión entre dos RLE's se lleva a cabo cuando están en renglones adyacentes y existe traslape en sus coordenadas x; un RLE se puede unir con un *BLOB*, si la coordenada y del RLE difiere en uno de alguna de las coordenadas, y_1 o y_2 del *BLOB* y cuando haya traslape en las coordenadas x; dos *BLOBs* pueden fusionarse cuando la coordenada y_1 de alguno de los *BLOBs* difiere en uno de la coordenada y_2 del otro y cuando las coordenadas x de ambos se traslapan.

Se utilizó el concepto de 8-conexidad para facilitar la fusión entre regiones. Cada pixel tiene ocho pixeles en su vecindad que son los contiguos a éste. Un pixel se conecta con otro del mismo color si se encuentra dentro de su vecindad.

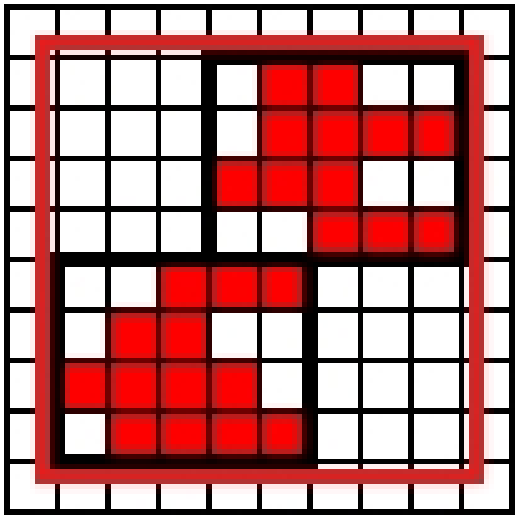

Figura 3.2: Unión de regiones en crecimiento

Cuando dos regiones R_1 y R_2 se unen (figura 3.2), se genera una región rectangular ($BLOB$) cuya diagonal es el segmento $\overline{p_1 p_2}$, donde

$$\begin{aligned} p_1 &= \{(x_1, y_1)\ en\ la\ imagen : x_1 = min\{x \in R_1 \cup R_2\} \wedge y_1 = max\{y \in R_1 \cup R_2\}\} \\ p_2 &= \{(x_2, y_2)\ en\ la\ imagen : x_2 = max\{x \in R_1 \cup R_2\} \wedge y_2 = min\{y \in R_1 \cup R_2\}\} \end{aligned} \tag{3.4}$$

El algoritmo que se utilizó para generar los *BLOBs* de cada uno de los colores, a partir de las regiones lineales (RLE's), fue el siguiente:

- **PARA** cada uno de los RLE's en la lista (RLEi)
- **SI** el RLE no esta incluido dentro de algún BLOB
- se crea un nuevo BLOB que contiene a este RLE
- **TERMINA SI**

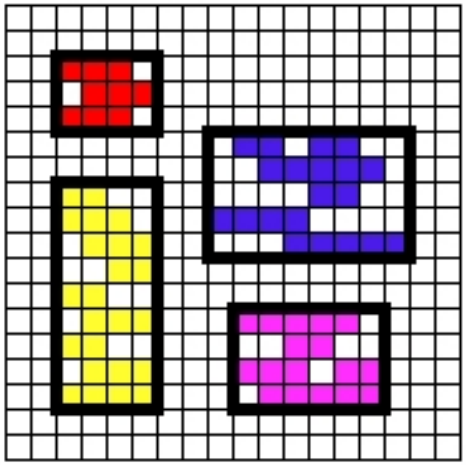

Figura 3.3: *BLOBs* generados a partir de la imagen

- **PARA** cada uno de los RLE's, empezando por el siguiente
- de RLEi (RLEj)
- se comparan las coordenadas de los RLE's (RLEi y RLEj)
- **SI** las coordenadas y difieren en uno y existe traslape en las
- coordenadas x
- **SI** RLEj no ha sido incluido en ningún BLOB
- se busca el BLOB en el que estaba incluido RLEi
- el BLOB crece de manera que abarca al RLEj
- **OTRO SI** el RLEj esta dentro de algún BLOB, pero no es el
- mismo en el que se encuentra RLEi
- se fusionan los *BLOBs* a los que pertenece cada uno de
- los RLE's
- **TERMINA SI**

- **TERMINA SI**
- **TERMINA PARA**
- **TERMINA PARA**

3.4.5. Eliminación de ruido

Una vez detectadas las regiones de color en la imagen, se procede a analizar cuáles de éstas corresponden a algún robot; hay que tomar en cuenta que la imagen capturada por la cámara puede llevar ruido y es probable que se hayan detectado regiones de color que no existen en la realidad. Por esta razón fue necesario implementar rutinas que permitieran eliminar *BLOBs* que sean identificados como ruido.

Primero se ordenan por área las listas de BLOBs. En la interfaz gráfica se integró un control para que el usuario defina el área mínima que debe tener un BLOB para no ser considerado como ruido. Una vez formadas las listas de *BLOBs* se ejecuta una rutina que las recorre y elimina todos los *BLOBs* cuya área sea menor que el valor dado por el usuario.

3.5. Detección de robots y orientación

El paso siguiente consiste en detectar a los robots y su orientación a partir de las listas ligadas que contienen las regiones de color de la imagen.

Se diseñó una configuración para los parches de colores que se colocarían arriba de los robots, como se muestra en la figura 3.4. El parche central debe ser color amarillo o azul, respetando las reglas de la liga *Small Size* de *RoboCup*.

Si el color del parche central representa al equipo contrario, no es necesario poner atención a los parches de alrededor; esto se debe a que cada equipo puede definir su propia configuración para los parches auxiliares y, por esta razón, resultaría muy complicado intentar calcular la orientación de los contrarios con un sistema de visión como el que se está desarrollando en el presente trabajo.

Si, por el contrario, el color del parche central resulta ser del equipo de los pumas, los parches de alrededor deberán seguir la configuración mostrada en la figura 3.4.

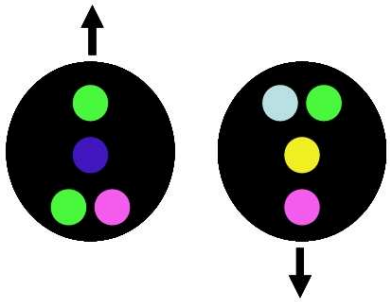

Figura 3.4: Configuración de parches de color en los robots

En la figura 3.4 se muestra el parche central (que puede ser azul o amarillo) rodeado por tres parches de otros colores acomodados en forma de un triángulo isósceles, cuya base es significativamente menor que su altura. El eje que va desde la base del triángulo (perpendicularmente) y llega hasta el vértice opuesto, define la dirección del robot (figura 3.5).

La combinación de colores de los tres parches que se encuentran alrededor del parche central, servirán como código para identificar la posición en la que juega el robot que se detecta (portero, atacante 1, atacante 2, atacante 3, atacante 4).

Para que el sistema de visión hiciera la detección de orientación utilizando esta configuración, fue necesario desarrollar un algoritmo que agrupara a cada uno de los parches con el color distintivo del equipo (amarillo o azul), con los tres parches de otros colores más cercanos a él.

El algoritmo resultante es el siguiente:

- **PARA** cada uno de los parches auxiliares detectados
- **PARA** cada uno de los parches centrales
- obtener la distancia entre el parche central y el parche auxiliar
- **SI** la distancia es la menor hasta el momento

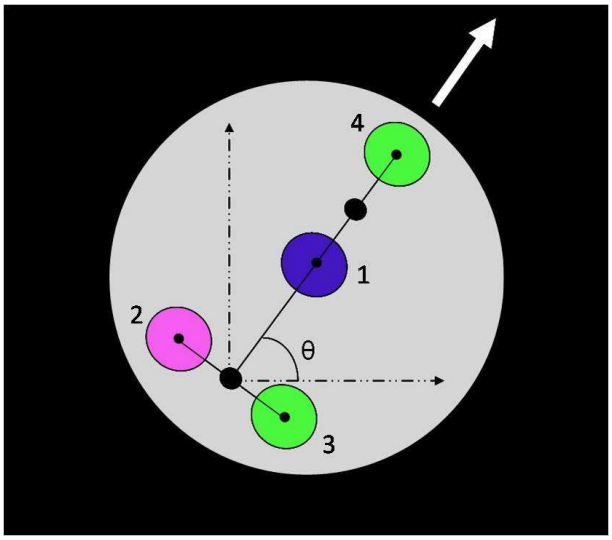

Figura 3.5: Obtención de la orientación del robot a partir de la configuración de parches

- se señala al parche central como posible más cercano
- **TERMINA SI**
- **TERMINA PARA**
- **SI** la distancia más pequeña es mayor que la distancia máxima permitida
- se descarta al parche auxiliar por estar muy lejos de cualquier
- parche central (ruido)
- **TERMINA SI**
- **OTRO**
- **SI** no se han encontrado aun los 3 auxiliares alrededor del
- central señalado
- se agrupan el parche auxiliar con el parche central
- (el mas cercano)

- **TERMINA SI**
- **PARA** cada parche auxiliar agrupado con el parche
- central señalado
- se comparan las distancias con el recién agrupado y se ordenan
- de menor a mayor
- **SI** hay mas de 3 auxiliares agrupados
- se descarta el que tenga la mayor distancia
- **TERMINA SI**
- **TERMINA PARA**
- **TERMINA OTRO**
- **TERMINA PARA**

Lo que hace el algoritmo anterior, es agrupar a los parches centrales con los parches auxiliares más cercanos. Al ir agrupando se ordenan por distancia, de manera que finalmente sólo se tomarán en cuenta los tres parches auxiliares más cercanos y se descartarán los demás; ésto ayuda a eliminar *BLOBs* que se hayan podido detectar por culpa del ruido en algún lugar de la cancha alejados de un parche central.

Una vez agrupados los parches, es necesario identificar de qué robot se trata. Para ello, se utilizó un código de colores de manera que cada uno de los robots del equipo tuviera un identificador único. Se definió el siguiente código:

Posición	Identificador	Color 1	Color 2	Color 3
Portero	1	Verde	Verde	Rosa
Atacante 1	2	Verde	Verde	Cian
Atacante 2	3	Verde	Rosa	Cian
Atacante 3	4	Rosa	Rosa	Verde
Atacante 4	5	Rosa	Rosa	Cian

Cuadro 3.1: Código de colores para identificar a cada robot.

El último paso en la detección de los robots es encontrar su orientación. Ya que se cuenta con los parches centrales y auxiliares agrupados, se utilizan sus posiciones para calcular su dirección. Como se observa en la figura 3.5 se calcula el punto medio entre el parche central (parche 1) y el parche 4 y el punto medio entre los parches 2 y 3. A partir de estos dos puntos (marcados en negro en la figura) se obtiene la pendiente de la recta que los une. Se calcula la tangente inversa de dicho valor y este es el ángulo de orientación del robot. En vez del primer punto se podría utilizar el propio centroide del parche 2, pero se tomo la decisión de utilizar el punto medio entre dos parches para minimizar el error.

3.6. Comunicaciones

El último módulo del sistema de visión es el de comunicaciones. En este punto, el sistema ya detectó a todos los robots (amigos y enemigos) y a la pelota, y corresponde enviar la información al módulo de inteligencia para que, a partir de estos datos, se pueda ordenar a los robots desempeñar alguna acción.

Se utilizaron *sockets* UDP para realizar dicha comunicación. Como se discute en el Marco Teórico, los *sockets* UDP tienen ventajas en esta aplicación por el hecho de que no se establece una conexión y, si un paquete se pierde en el camino, el receptor no quedará en espera de él hasta que lo reenvíen. En el caso de este sistema, si se pierde un paquete, conviene olvidarse de él y tomar el siguiente paquete, ya que de la otra manera, la inteligencia tendría que quedar en espera, además de que recibiría datos obsoletos y pondría en cola de espera a los paquetes que sí son actuales.

Para manejar estos *sockets* se desarrolló una clase (myUDPSockets) para encapsular las funciones de la librería *winsock.h* que implementa un cliente UDP. Se decidió que el sistema de visión funcionaría como cliente, debido a que el sistema de planeación/navegación deberá ser el servidor, ya que recibirá también las conexiones de las *PocketPC* de todos los robots.

Esta clase tiene como atributos principales el manejador del *socket*, un *buffer* para almacenar el mensaje que se envía, otro buffer para almacenar el nombre o dirección IP del servidor (host) al que se va a conectar y el puerto en el que va a trabajar. La clase UDPSockets ofrece un método para crear el *socket* y establecer los parámetros (host y puerto), y otro para enviar mensajes utilizando el *socket*.

Capítulo 4

Desarrollo del sistema de planeación/navegación

4.1. Arquitectura *híbrida (deliberativa/reactiva)*

En este trabajo se utilizó el paradigma *híbrido* para el desarrollo del sistema de planeación/navegación. Se tiene un solo sensor que indica en todo momento la posición del robot, la de los robots enemigos y la de la pelota; éste es el resultado del procesamiento de video. Con base en esta información, el planeador calcula el punto a donde deberá moverse el robot; posteriormente se utiliza el algoritmo de *campos potenciales* para poder navegar de manera reactiva hacia el punto señalado por el planeador. Si el comportamiento de *campos potenciales* detecta un robot enemigo cercano, deberá evadirlo y seguir su trayectoria.

4.2. Diseño del sistema

El sistema de planeación/navegación (Director Técnico) fue desarrollado en C++ .NET utilizando programación orientada objetos. Se diseñó una clase *Robot* que implementa todos los métodos que necesitan los robots para funcionar y contiene los atributos más importantes:

- Identificador de robot: Indica si es amigo o enemigo. Si es amigo también indica de que robot amigo se trata
- Tipo de robot: Indica si es portero o atacante
- Posición: Indica la posición del robot

- Indicador de activo: Indica si el robot fue detectado por la visión o no

Para cada uno de los robots que se encuentren jugando, habrá un objeto de la clase *Robot*. Asimismo, se diseñó una clase *Amigo* que es subclase de la clase *Robot*. Esta clase hereda las características básicas de un robot. La clase amigo cuenta con atributos y métodos que sólo necesitan los robots amigos:
Atributos:

- Ángulo: dirección hacia donde apunta el frente del robot
- *Socket*: estructura *socket* para realizar la conexión con la *PocketPC* que va montada en el robot físico

Métodos importantes:

- *Campos potenciales*: implementa el algoritmo de *campos potenciales*, como función de movimiento para los robots amigos

La clase *Amigo* tiene, a su vez, dos subclases que heredan sus métodos y atributos: *Portero* y *Atacante*. La diferencia esencial entre estas tres clases es que cada una de ellas implementa un método de inteligencia diferente. Es en el método inteligencia donde se define la estrategia para cada una de las posiciones de juego (portero o atacante).

4.3. Planeación

Como se mencionó anteriormente, se diseñaron dos comportamientos diferentes, según la posición que juega el robot, implementados en las clases *Portero* y *Atacante*. Los dos tipos de robot, utilizan el mismo método de navegación, por *campos potenciales*, pero es el planeador quien decide los puntos de atracción y repulsión, según la posición que juegue cada robot.

4.3.1. Atacante

Habrá cuatro robots de este tipo en la cancha. Cada uno de los robots, deberá tener un área sobre la cual desempeñarse, para evitar que los robots del mismo equipo se estorben o que haya colisiones entre ellos. Para estos fines, la cancha se divide

imaginariamente en cuatro regiones iguales, en cada una de las cuales actuarán los robots atacantes.

La posición inicial o de descanso de los robots, será el centro de su región. El robot permanecerá en esta posición mientras la bola este fuera de su región. En cuanto la pelota ingrese en su área, el robot deberá moverse e ir hacia ella. En cuanto el robot se encuentre a una distancia de tiro de la pelota, activará el disparador y irá hacia ella para patearla.

Es importante evitar tiros hacia la portería propia, por lo que el se implementó una rutina para evitar dicha situación:

- **REPITE**
- **SI** el robot apunta hacia su propio lado
- se traza la recta que va desde la portería enemiga a la pelota
- el atractor es un punto en la prolongación de esta recta,
- a 10 cm de la pelota (figura 4.1)
- **OTRO**
- **SI** el robot está dentro del arco con vértice en el centro de la portería
- enemiga y que abre en dirección de la pelota (figura 4.2)
- se usa como atractor la pelota
- **SI** la distancia entre el robot y pelota es de 10 cm o menos
- se enciende el disparador
- **TERMINA SI**
- **TERMINA SI**
- **TERMINA OTRO**
- **TERMINA REPITE**

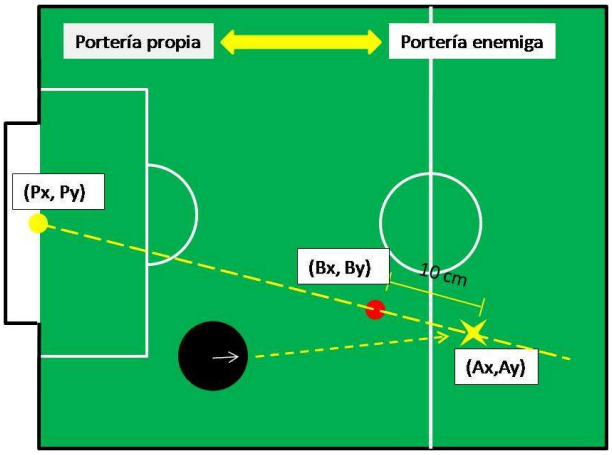

Figura 4.1: Elección del atractor cuando el robot apunta hacia su portería

En las figuras 4.1 y 4.2 se observan los puntos que se utilizan como atractores dependiendo de la posición del robot y la pelota.

Para obtener las coordenadas del atractor (el punto (A_x, A_y) de la figura 4.1) se utilizan las siguientes ecuaciones:

Ecuación de la recta:

$$\begin{aligned} m &= \frac{P_y - B_y}{P_x - B_x} \\ b &= P_y - mP_x \\ y &= \frac{P_y - B_y}{P_x - B_x}x + P_y - mP_x \end{aligned} \tag{4.1}$$

Obtención de las coordenadas del punto A:

$$\begin{aligned} \Delta y &= m\Delta x + b \\ \Delta x^2 + \Delta y^2 &= 10^2 \\ \Delta x &= \sqrt{10^2 - \Delta y^2} \\ \Delta x &= \sqrt{10^2 - m\Delta x + b^2} \\ A_x &= B_x + \Delta x \\ A_y &= B_y + \Delta y \end{aligned} \tag{4.2}$$

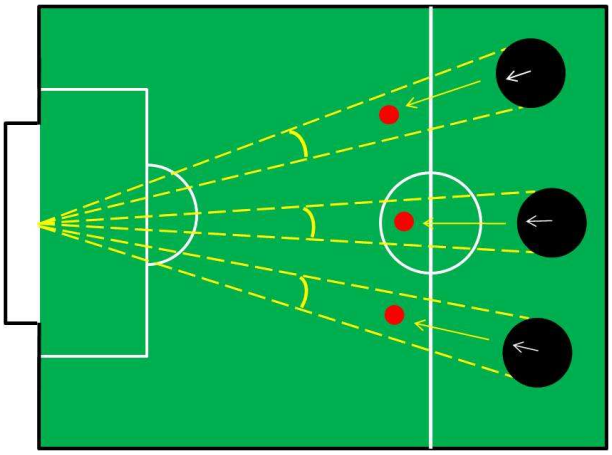

Figura 4.2: Posición correcta para tiro a gol

En la figura 4.3 se observa como se obtiene el arco de tiro a partir de la recta obtenida anteriormente. El ángulo α es el ángulo que tiene la recta obtenida anteriormente con respecto al eje x; el ángulo β es el tamaño del arco en el que el robot estará en posición de tiro; el ángulo θ es el ángulo que forma la recta que une al robot con el centro de la portería.

$$\begin{aligned} \alpha &= arctan(m) \\ \gamma_1 &= \alpha - \frac{\beta}{2} \\ \gamma_2 &= \alpha + \frac{\beta}{2} \\ \theta &= arctan\left(\frac{R_y - P_y}{R_x - P_x}\right) \end{aligned} \tag{4.3}$$

Para ver si el robot se encuentra en posición de tiro, basta con ver si el ángulo θ del robot, se encuentra entre γ_1 y γ_2.

El ancho del arco β es una variable que se fija en tiempo de ejecución, de manera que se pueda calibrar la puntería de los robots.

4.3.2. Portero

El portero siempre deberá permanecer a una corta distancia de la portería y tratar de obstruir el paso de la pelota. Por esta razón se decidió que este jugador siempre

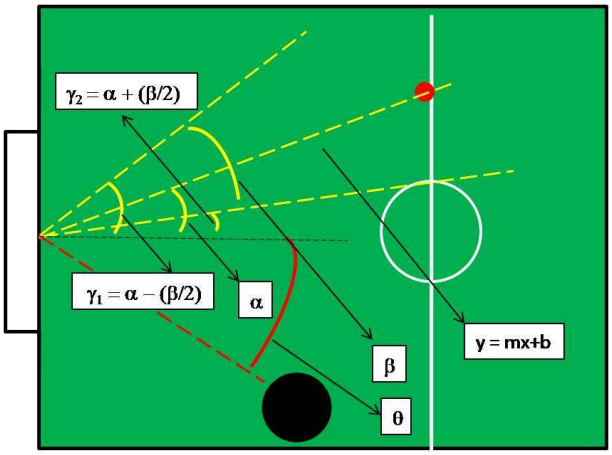

Figura 4.3: Obtención del arco de tiro

deberá encontrarse sobre una línea imaginaria a 15cm de la línea de meta. El punto exacto es la intersección de esta línea con una recta ($y(x)$) que va desde el centro de la portería a la posición de la pelota; este es el atractor que utiliza el algoritmo de campos potenciales. De esta manera, se bloquearán los tiros que vayan al centro de la portería. Una vez obtenido el punto de atracción, el planeador invocará al navegador (*campos potenciales*) y le enviará como atractor el punto que se obtuvo y como repulsor, las posiciones los robots amigos y enemigos (para evitar colisiones). Asimismo, cuando se detecte que la pelota se encuentra a una distancia de 15 cm o menos del portero, éste deberá encender el pateador, para repelerla.

Dado que no hay manera de detectar la orientación de los robots enemigos resulta imposible, en este punto del proyecto, saber la dirección a donde va dirigida la pelota cuando un enemigo dispara. Por esta razón se considera que el lugar donde será más probable detener los tiros de los contrarios es en la recta que une a la pelota con el centro de la portería.

En la figura 4.4 se observa la manera en que se obtiene el atractor para el portero; a continuación se desarrollan las ecuaciones necesarias:

La recta y que se muestra en la figura 4.4, se obtiene de la misma manera que se hizo en el caso de los robots atacantes (ecuaciones 4.2).

El punto de intersección de dicha recta con la línea imaginaria que está a 15cm

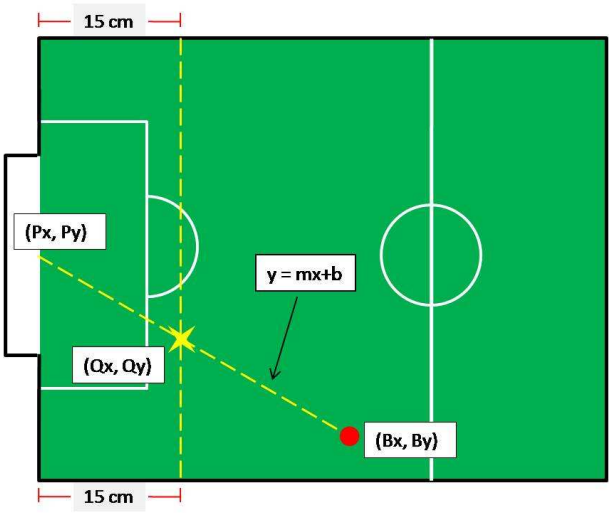

Figura 4.4: Obtención del punto atractor para el portero

de la línea de meta se obtiene haciendo la variable $x = 15$:

$$\begin{aligned} x &= 15 \\ y &= mx + b \\ y &= 15m + b \\ Q = (Q_x, Q_y) &= (15, 15m + b) \end{aligned} \tag{4.4}$$

Donde el punto Q es el atractor que se enviará al navegador (*campos potenciales*).

4.4. Navegación: *Campos potenciales*

El comportamiento más importante que se desarrolló en el sistema de planeación/navegación, fue el de *campos potenciales*. Éste es el comportamiento de más bajo nivel implementado por el Director Técnico, y será utilizado por comportamientos de más alto nivel, como función de movimiento.

Dado que el ambiente donde se desenvuelven estos robots es muy controlado, se logró simplificar la representación del mundo que utiliza el robot para funcionar. No se necesita una representación global del mundo, dado que los robots y la pelota pueden tomarse como un punto en el espacio y que no hay obstáculos fijos. Entonces,

la única representación del mundo que se tiene es la lista de posiciones de todos los robots y la posición de la pelota.

Dentro de la clase Robot se implementó un método que implementa los *campos potenciales*, que no necesita un mapa del mundo, sino únicamente un arreglo de puntos repulsores y un punto atractor. Dado que es un ambiente muy dinámico en el que los obstáculos son los robots enemigos y cambian de posición constantemente, es necesario calcular la fuerza de repulsión que siente el robot a causa de los enemigos en cada instante de tiempo; no se puede guardar una matriz con los vectores de repulsión.

El método implementado calcula un vector hacia donde deberá moverse el robot ($\bar{F}$). Para ello, calcula la fuerza que siente el robot hacia el punto definido como atractor ($\bar{F_A}$) y luego calcula la fuerza de repulsión ($\bar{F_R}$) que siente por efecto de todos los repulsores que se encuentren a una cierta distancia máxima d_0.

Sea P la posición, del robot, P_A la posición del atractor y P_{R_i} la posición del i-ésimo repulsor.

$$\bar{F_A} = \frac{\epsilon}{|\overline{PP_A}|}\left(\bar{P_A} - \bar{P}\right) \tag{4.5}$$

$$\bar{F}_{R_i} = n\left(\frac{1}{|\overline{PP_{R_i}}|} - \frac{1}{d_0}\right)\left(\frac{1}{|\overline{PP_{R_i}}|^2}\right)\left(\bar{P} - \bar{P_{R_i}}\right) \tag{4.6}$$

$$\bar{F_R} = \sum \bar{F_{R_i}} \tag{4.7}$$

$$\bar{F} = \bar{F_R} + \bar{F_A} \tag{4.8}$$

Las constantes ϵ y n se calibran de manera empírica para calibrar el funcionamiento de los *campos potenciales*, ajustando los efectos de las fuerzas de atracción y repulsión. d_0 es la distancia máxima a la que se puede sentir el efecto repulsivo de un objeto.

El método desarrollado regresa el vector de fuerza resultante.

4.5. Comunicaciones y *threads*

En el sistema de planeación/navegación, nuevamente fue necesario desarrollar un módulo de comunicaciones con *sockets* para poder recibir la información proveniente del sistema de visión y, después de procesarla, enviar los comandos a las *PocketPC's*

de los robots.

Nuevamente se utilizaron *sockets* UDP que no establecen conexión, de manera que si se pierden paquetes en el camino, el extremo receptor no queda en espera de su reenvío.

Para implementar estos *sockets* se utilizó una clase previamente desarrollada en C++.NET en el laboratorio de BioRobótica (UDPSockets). Esta clase encapsula todas las funciones de conexión, escritura y lectura de un *socket* UDP. Fue necesario hacer modificaciones a esta clase, para evitar que hubiera excepciones cuando el usuario decidía terminar el programa mientras el *socket* se encontraba abierto. Para lograrlo, simplemente se mejoró el manejo de las excepciones propias de los *sockets*.

Este programa hace uso intensivo de las comunicaciones vía *sockets*. Cuando los cinco robots están jugando, es necesario que se abran cinco puertos UDP para comunicarse con ellos, y dos puertos más para comunicarse con los dos servidores de visión que se utilizan cuando se juega en una cancha completa. En total son siete conexiones UDP que se deben manejar todas simultáneamente. Esta situación requiere que se planteé una solución que permita realizar dicha tarea.

Dado que el sistema necesita que se calculen en todo momento las jugadas, al mismo tiempo que se recibe y envía información a los siete *sockets* abiertos y que el usuario manipula la interfaz gráfica, fue necesario hacer uso de *threads* (*hilos*). Es muy importante que el usuario pueda modificar, en tiempo de ejecución, variables del programa como son las constantes de los campos potenciales, el arco de tiro, etc., o que se pueda en todo momento detener el juego o manejar las conexiones con los robots individualmente.

Por estas razones, se levantaron dos *threads*, uno de los cuales se encarga de las comunicaciones y cálculos y el otro del manejo de la interfaz visual. De esta manera, el *thread* que controla la interfaz gráfica puede modificar los valores de las variables antes mencionadas para que el otro *thread* las incorpore de inmediato a sus cálculos.

Capítulo 5

Pruebas y Resultados

Una vez terminado el desarrollo del sistema, se diseñaron pruebas diferentes en las que se pone a prueba el sistema completo (considerando como sistema completo al lazo cerrado que se forma entre los robots, el sistema de visión y el Director Técnico).

5.1. Prueba 1: ir hacia la pelota y evadir obstáculo

El objetivo de esta prueba es probar a fondo el algoritmo de campos potenciales, al ordenarle al robot que se mueva desde una posición (siempre fija) hacia la pelota que se encuentra fija en otra posición. En la recta que une al robot y la pelota se coloca un robot enemigo que funge como obstáculo al que el robot debería evadir (figuras 5.1 y 5.2).

En esta prueba, se pretende también encontrar la distancia mínima de repulsión y la relación entre las constantes ϵ y η que hagan que el robot siga una trayectoria óptima hacia la pelota, en el menor número de movimientos posibles.

5.1.1. Método

- Se colocan el robot de prueba y la pelota en la cancha, en la misma posición en cada repetición.
- Se coloca un robot enemigo en el punto medio entre la pelota y el robot de prueba.
- Se enciende el sistema completo (robot, visión y Director Técnico).

Figura 5.1: Prueba 1. Colocación de los robots y la pelota

- Se cuenta el número de movimientos en los que el robot logró alcanzar la pelota o se indica si hubo colisión.
- Se repitió esta prueba hasta 30 veces para cada valor de las constantes ϵ y η que se quieren probar.

5.1.2. Resultados

Esta prueba arrojó los siguientes resultados:

Serie	D_{RR}	D_{RP}	η/ϵ	d. mín.	rep.	# col.	% col.	mov. prom.	$\sigma(\%)$
1	5	5	2.5	30cm	30	3	10 %	28.90	17.93 %
2	5	2	2.5	30cm	30	3	10 %	73.36	29.67 %
3	5	5	1.5	60cm	30	0	0 %	24.67	10.12 %
4	5	2	1.5	60cm	5	0	0 %	> 100	—
5	5	5	0.6	60cm	30	1	3.33 %	12.07	9.95 %
6	5	2	0.6	60cm	30	2	6.67 %	12.33	11.54 %

Cuadro 5.1: Resultados de la prueba 2

- Serie: número de la serie de repeticiones.
- D_{RR}: distancia del robot amigo al robot enemigo (figura 5.2).

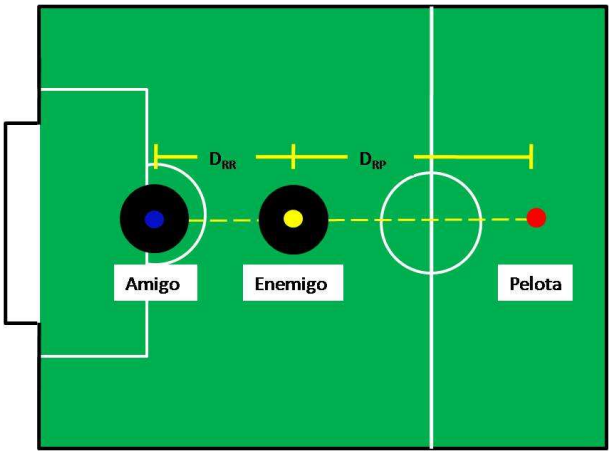

Figura 5.2: Prueba 1. Colocación de los robots y la pelota

- D_{RP}: distancia del robot enemigo a la pelota (figura 5.2).
- η/ϵ: relación entre las constantes de los campos potenciales (ponderación entre fuerzas repulsivas y atractivas).
- d. min.: radio de acción de los campos potenciales.
- rep.: número de veces que se repitió la prueba.
- # col.: número de veces que el robot amigo chocó contra el robot enemigo.
- % col.: porcentaje de veces que el robot amigo chocó contra el enemigo.
- mov. prom.: número promedio de movimientos que utilizó el robot para llegar a la pelota.
- $\sigma(\%)$: desviación estándar porcentual del número de movimientos.

A continuación se analiza lo presentado en la tabla 5.1:

- $\eta/\epsilon = 2.5$, d. mín.$= 30cm$:

  - $D_{RR} = 5, D_{RP} = 5$:
    Esta prueba se desarrolla exitosamente, aunque el número de movimientos promedio para llegar a la pelota es relativamente alto. El número de colisiones es muy aceptable.

- $D_{RR} = 5, D_{RP} = 2$:
  En esta prueba, el número de movimientos promedio es extremadamente alto y, por lo tanto, fue necesario buscar otro juego de parámetros que mejoraran el desempeño.
  La constante de repulsión resulta muy alta en comparación con la de atracción, y el robot queda atrapado en un mínimo local en el que es atraído por la pelota y repelido con más fuerza por el robot enemigo. A pesar de ésto, el robot, después de muchos intentos, logra llegar hasta la pelota debido a que el radio de acción de los campos potenciales es pequeño.

- $\eta/\epsilon = 1.5$, d. mín.$= 60cm$:

  - $D_{RR} = 5, D_{RP} = 5$:
    En esta prueba, el número de movimientos promedio mejora con respecto a la prueba anterior. Esto se debe a que la constante de repulsión es más baja en relación a la de atracción, al mismo tiempo que el radio de acción de los campos potenciales se amplía. De esta manera, el robot empieza a sentirse repelido antes y comienza a rodear al obstáculo con anticipación, trazando una mejor ruta y con menos riesgos de caer en mínimos locales.
  - $D_{RR} = 5, D_{RP} = 2$:
    De la misma manera como sucedió en la prueba anterior, el robot cae en un mínimo local. La diferencia es que, como el radio de acción de los campos potenciales es mayor, nunca logra salir y alcanzar la pelota. Esta prueba no se llegó a repetir las 30 veces que se habían establecido, dado que, cuando el número de movimientos es mayor a cien se considera como un fracaso. Después de varias repeticiones en las que siempre fracasó, se optó por suspender la prueba y descartar las constantes propuestas.

- $\eta/\epsilon = 0.6$, d. mín.$= 60cm$:

  - $D_{RR} = 5, D_{RP} = 5$:
    En esta prueba se tiene un promedio de movimientos excelente, tomando en cuenta la distancia a la que se encuentra el robot amigo de la pelota. El número de colisiones también mejora con respecto a la primera prueba y resulta un número muy aceptable. La mejora en el resultado se debe a que se tiene un radio de acción grande para los campos potenciales, pero la

constante de repulsión es más baja con respecto a las demás pruebas. Esto hace que el robot empiece a desviarse para evitar al robot enemigo con anticipación, generando una trayectoria que requiere menos pasos pero, al mismo tiempo, la repulsión del enemigo no evita que el robot avance con determinación hacia la pelota.

- $D_{RR} = 5, D_{RP} = 2$:
  El resultado de esta prueba es muy bueno. El número de pasos en el que llega el robot a la pelota es realmente pequeño, considerando los problemas encontrados en las pruebas anteriores. El éxito de esta prueba se debe a que la constante de repulsión es menor a la de atracción y esto evita que se caiga en mínimos locales. Se podría pensar que esto favorecería a las colisiones, pero hay que considerar que mientras más cerca se encuentre el robot de un obstáculo, más grande es la fuerza repulsiva (y ésta crece cuadráticamente con el decremento de la distancia: ecuación 4.5).

En la figura 5.3 se muestra una aproximación a las trayectorias que describió el robot en cada una de las pruebas. Las imágenes que muestran un tache indican que, en la serie que corresponde, el robot cae en un mínimo local del que tarda muchos movimientos en salir, o bien, nunca sale.

Esta prueba arroja, como resultado final, que los parámetros óptimos encontrados para utilizar en el algoritmo de campos potenciales para la navegación del robot son:

- $\eta/\epsilon = 0.6$
- d. mín. $= 60cm$

5.2. Prueba 2: seguir pelota y evitar colisión en ambiente dinámico

La segunda prueba no es cuantificable y consistió en probar el algoritmo de campos potenciales en un ambiente dinámico, primero cuando sólo hay un atractor y ningún repulsor y posteriormente cuando se tiene un atractor y un repulsor. Para ello se ordena al robot seguir la pelota en movimiento.

5.2.1. Método parte 1

- Se coloca a un robot y la pelota en la cancha.

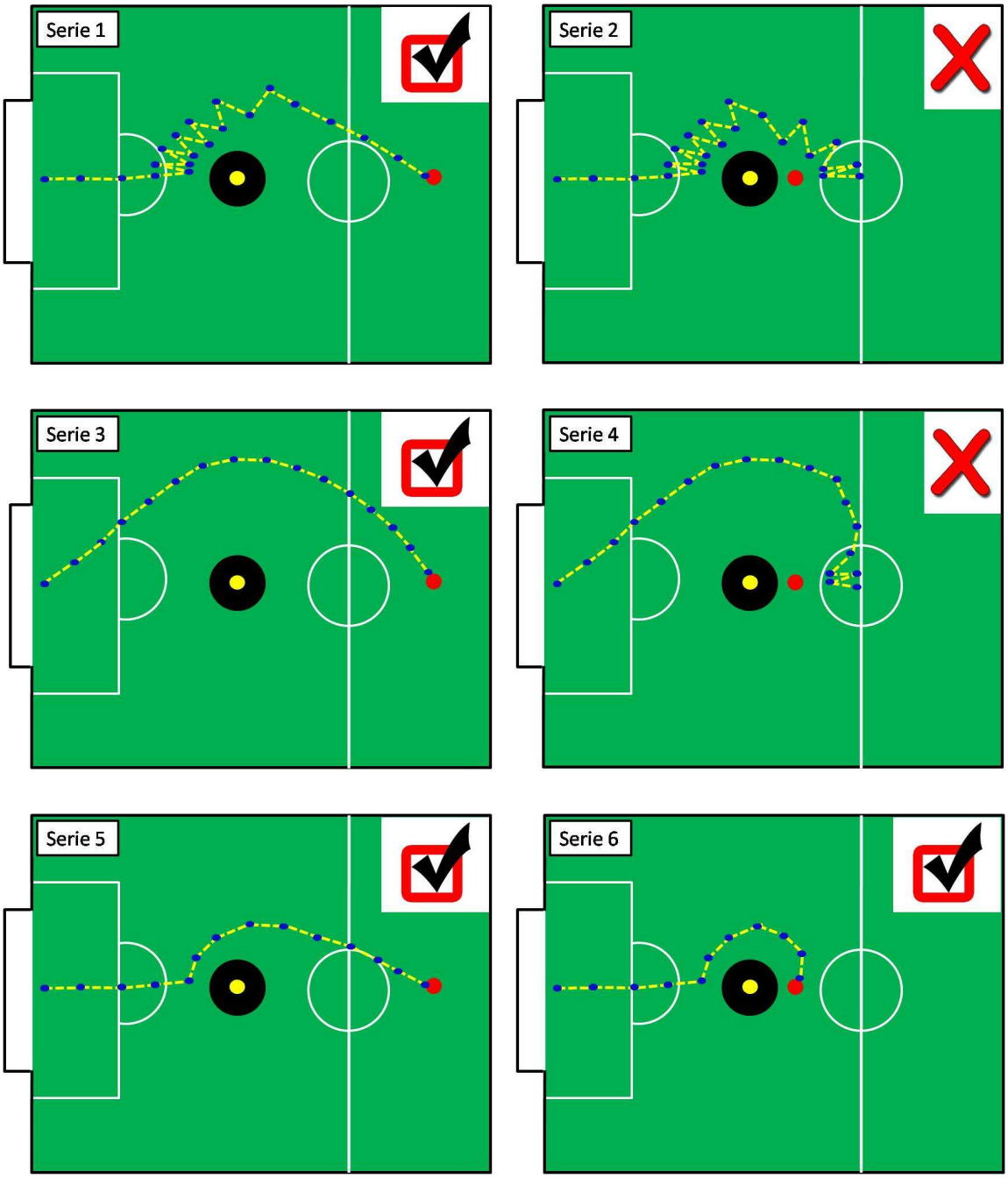

Figura 5.3: Trayectorias del robot para la prueba 1

- Se enciende el sistema completo (robot, visión y Director Técnico).
- Una persona mueve la pelota utilizando un palo de golf (u otro objeto largo que no interfiera con la visión).
- Se observa durante períodos de por lo menos cinco minutos el comportamiento del robot.

5.2.2. Método parte 2

- Se coloca a un robot amigo, a un enemigo y la pelota en la cancha.
- Se enciende el sistema completo (robot, visión y Director Técnico).
- Una persona mueve la pelota utilizando un palo de golf (u otro objeto largo que no interfiera con la visión).
- La persona mueve esporádicamente al robot enemigo.
- Se observa durante períodos de por lo menos cinco minutos el comportamiento del robot.

5.2.3. Resultados

Desde que se le da la orden al robot de que siga a la pelota, este comienza a moverse siempre en dirección de la misma. Si la pelota cambia de dirección, el robot rápidamente cambia de dirección para ir hacia ella. Si la pelota llega a quedarse quieta durante algunos instantes, el robot le da alcance.

Cuando se agrega a un robot enemigo en la prueba, el robot propio intenta ir por la pelota, pero cuando llega a encontrarse al robot enemigo en su camino peligrosamente cerca, cambia la dirección para evitar la colisión. En ocasiones, cuando el obstáculo queda en la trayectoria hacia la pelota en movimiento, el robot se pierde de su camino durante algunos instantes, hasta que la pelota llega a alguna posición en la que el robot puede librar al oponente y seguir avanzando hacia ella.

5.3. Prueba 3: tiro a gol

Con esta prueba se pretende observar un comportamiento más complejo que los anteriores, que es el tiro a gol. La primera parte de esta prueba fue colocar al robot

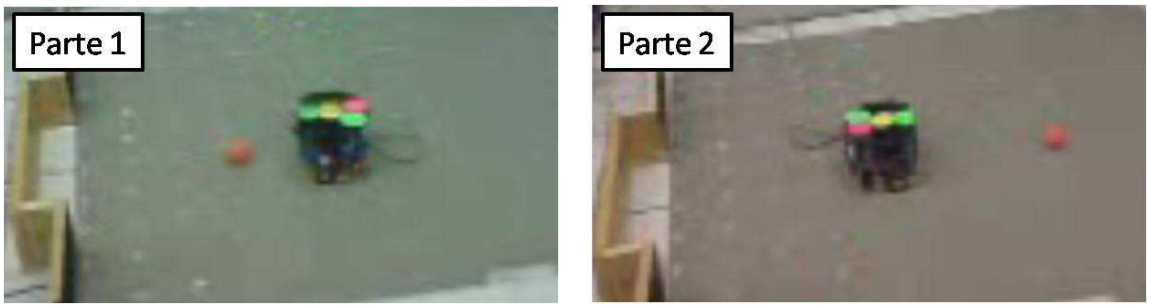

Figura 5.4: Prueba 3. Colocación del robots y la pelota con respecto a la portería enemiga

en línea reta con la pelota y la portería enemiga de manera que sólo tuviera que avanzar y tirar. En la segunda fase, se colocó al robot en diferentes regiones de la cancha viendo hacia su propia portería y sin estar en línea con la pelota y la portería contraria. Se pretende que el robot utilice su comportamiento de atacante, rodee la pelota y tire a gol hacia el lado correcto.

5.3.1. Método

Parte 1

- Se colocan el robot de prueba y la pelota en cualquier punto de la cancha en línea con el centro de la portería enemiga (figura 5.4 izquierda).
- Se ordena al robot que comience a jugar.
- Se repitió esta prueba 90 veces.

Parte 2

- Se colocan el robot de prueba y la pelota en algún punto de la cancha en el que no estén en línea con el centro de la portería enemiga (figura 5.4 derecha).
- Se ordena al robot que comience a jugar.
- Se repitió esta prueba 90 veces.

5.3.2. Resultados

Los resultados de la prueba 3 se muestran en la siguiente tabla:

Parte	# de tiros	# anotaciones	% de anotaciones
1	90	59	65.55 %
2	90	65	72.22 %

Cuadro 5.2: Resultados de la prueba 3

Considerando la complejidad del problema del fútbol para robots móviles, se considera que el resultado de la prueba 3 (tabla 5.2) es un éxito. El robot logra en aproximadamente 7 de cada 10 ocasiones, acomodarse en la posición correcta y patear la pelota en dirección de la portería enemiga.

5.4. Observaciones

Durante el desarrollo de las pruebas, fue necesario enfrentar ciertos problemas que fueron surgiendo:

El sistema completo utiliza varias conexiones con *sockets* UDP: dos entre el sistema de visión y el sistema de planeación/naegación y cinco entre el sistema de planeación/navegación y los robots. La velocidad de la transmisión de datos resulta muy importante, dado que de ello depende el tiempo de respuesta del robot ante las situaciones de su entorno. Se observó que si se utiliza el *access point* que se tiene en el laboratorio que conecta a todas las computadoras con internet, la transmisión de información entre los diferentes puntos del sistema a veces se vuelve lenta y cortada. Una posible solución a este problema era contar con un *access point* dedicado exclusivamente a las pruebas de *RoboCup*, pero desafortunadamente en el laboratorio de BioRobótica no se cuenta con un *access point* sobrante. La solución que se dio para este problema, fue levantar el sistema de visión y el Director Técnico en la misma PC y hacer conexiones punto a punto (ad-hoc) entre dicha PC y la *PocketPC* del robot que estuviera a prueba. Esto resolvió el problema y mejoró en gran medida la comunicación entre las partes del sistema.

Cuando comenzó el proyecto de la *RoboCup Small Size* en el laboratorio de BioRobótica, se desarrollaron unos robots con una electrónica y una programación (de los PICs) que cumplían con las necesidades del proyecto en ese momento. El sistema

de control de los robots fue hecho empíricamente, basándose en una tabla que relaciona el ángulo de giro del robot con la cantidad de pulsos que se le deben enviar a los motores; esto trae consigo que el giro que logra realizar el robot para el número de pulsos que se le mande a los motores depende de la carga restante de la batería. Eventualmente cambió la electrónica de potencia pero no cambió la programación de la tabla usada para el control; de esta manera, la tabla quedó obsoleta y los valores de giro y avance que se envían, no corresponden al movimiento que realmente realizará el robot. Además se pudo observar que para cada superficie en donde se probaba el robot (i.e. alfombra, piso de loseta, fomi, etc.) cambiaba la forma de girar y avanzar. Para solucionar este problema, fue necesario incluir en la interfaz gráfica del Director Técnico, una variable que se utilizaría como un factor de corrección del ángulo de giro y una variable que ajustaría la distancia que avanza el robot en cada movimiento. De esta manera, ahora se puede calibrar estos parámetros y que el robot gire y avance bien en cada superficie donde se ponga a prueba.

A pesar de la corrección implementada al giro y avance del robot, en cada movimiento se tiene un porcentaje de error muy alto (tanto en el ángulo de giro como en la distancia de avance). Inclusive (por problemas de mecánica) cuando se le pide al robot avanzar en línea recta después de girar, este curva su trayectoria (por la inercia que lleva del giro). En un principio se pensó que este problema impediría obtener algún resultado de las pruebas pero, sorprendentemente, se observó que el algoritmo de campos potenciales subsana y corrige este error en cada iteración. Dado que es una prueba dinámica en la que se tiene un lazo cerrado entre el robot, la visión y el sistema de planeación/navegación, el error en el movimiento del robot que se tiene en cada paso, lo detecta la visión, y el algoritmo de campos potenciales intenta corregirlo en el movimiento siguiente. De esta manera, a pesar del gran error que genera la falta de un sistema de control, se pudieron llevar a cabo las pruebas exitosamente, como se reporta anteriormente.

En general fue problemático llevar a cabo las pruebas, dado que constantemente surgían errores con los robots. En su mayoría eran problemas electrónicos (falsos contactos, cortos circuitos, integrados quemados), aunque también se presentaron errores en la configuración de las conexiones inalámbricas y errores mecánicos (ruptura de alguna pieza). Esto hizo que este proceso fuera más lento de lo que se había previsto.

Capítulo 6

Conclusiones

6.1. Dificultades en el desarrollo

La robótica es un tema interdisciplinario, por lo que, en un proyecto en esta área, se requiere de la participación de personas especializadas en cada una de las disciplinas que la conforman.

En el caso particular del proyecto que se desarrolló en la presente tesis, esto resultó ser un problema serio. El proyecto comenzó con unos prototipos de robots diseñados exclusivamente por ingenieros electrónicos e ingenieros en computación, sin el apoyo de ingenieros mecánicos ni gente especializada en control.

Más tarde, se pudo contar con la colaboración de gente especializada en ingeniería mecánica y se construyó el segundo prototipo (el actual); la etapa de control, sin embargo, sigue siendo la primera que se desarrolló, sin la participación de gente del área de control y tiene muchas limitaciones.

Este control fue desarrollado elaborando una tabla que relaciona el número de pulsos que reciben los motores con el ángulo que el robot gira o con la distancia que avanza. Este sistema de control falla cuando cambia ligeramente la fricción del piso o cuando la carga de las baterías de los robots esta baja.

Además, en las competencias de *RoboCup* recientes, los robots que se utilizan ya no usan tracción diferencial, sino tres o cuatro ruedas omnidireccionales, con tres o cuatro motores respectivamente, para controlarlas, que dotan al robot de la capacidad de moverse en cualquier dirección sin necesidad de girar previamente. Esto

obliga a que se desarrolle un tercer prototipo de los robots para poder tener un nivel competitivo; aquí el proyecto se enfrenta al problema adicional de que, aun teniendo los recursos, la adquisición de equipo y materiales, por parte del laboratorio, en ocasiones tiene una demora, por cuestiones administrativas, que rebasa los tiempos del proyecto y entorpece su desarrollo.

Actualmente, en el Laboratorio de Biorobótica hay personas trabajando en el área de visión, inteligencia artificial y microcontroladores, pero para el desarrollo a futuro del proyecto, será necesario contar también con personas especializadas, por lo menos, en las áreas de control y mecánica.

Otra dificultad que se tiene por el hecho de que el proyecto sea interdisciplinario es que se requiere de un equipo de trabajo muy numeroso en donde la logística se vuelve complicada. Cuando hay que coordinar el trabajo de varias personas para el desarrollo del proyecto, es fácil que la demora de una persona al desarrollar el módulo que le corresponde, afecte el avance de otras partes del proyecto que requieren del funcionamiento de dicho módulo.

6.2. Logros

A pesar de las dificultades señaladas en la sección anterior, se desarrollaron satisfactoriamente un sistema de visión y un sistema de planeación y navegación que cumplen con las expectativas planteadas en los objetivos.

El sistema de visión resulta ser muy eficiente dado que puede operar con webcams para las antiguas reglas de la competencia, en donde la cancha tenía una medida de 4.9m x 3.4m. Esto representa un gran logro del proyecto, dado que la mayoría de los equipos en otros piases, utilizan cámaras digitales de mucha más alta tecnología, que cuentan con mayor resolución, pero cuyo costo es varias veces más elevado que el de las webcams.

Por otro lado, el sistema de planeación/navegación, logró que los robots adoptaran los comportamientos definidos para el juego de fútbol. La arquitectura planteada para dicho sistema contempla que el ambiente es muy dinámico en su parte reactiva, pero también la necesidad de que se pueda seguir una estrategia de juego a largo plazo en su parte deliberativa o jerárquica. Se observa como la parte reactiva (implementada por los campos potenciales) reacciona de inmediato ante los cambios en

las posiciones de los robots enemigos y la pelota, mientras que el objetivo principal del robot (por ejemplo ir hacia la pelota) sigue fijo hasta que se completa con éxito o cambian las condiciones del juego.

El algoritmo de campos potenciales resultó ser una elección muy acertada para encargarse de la navegación, dado que logró corregir los errores en el movimiento de los robots, generados por el sistema de control. Dado que el sistema está a lazo cerrado y que es muy dinámico, cuando el sistema de control comete errores (y hace que el robot se mueva con un alto grado de incertidumbre) el sistema de visión detecta la nueva posición del robot, y los campos potenciales (que se encuentran dentro del sistema de planeación/navegación) tratan de corregir el error al generar el siguiente comando para el robot.

En general, el gran problema del algoritmo de campos potenciales es la posibilidad de caer en mínimos locales y que el robot nunca alcance su objetivo; sin embargo, para el proyecto de la *RoboCup*, el ambiente en el que se desempeñan los robots es muy dinámico y los posibles obstáculos para un robot son sólo los demás robots que se encuentran en la cancha que fueron considerados como un repulsor puntual. Esto da como resultado que sean muy pocas las ocasiones en las que el robot cae en un mínimo local y cuando esto llega a ocurrir, el robot no tarda en salir de dicha situación.

El sistema de planeación/navegación fue desarrollado utilizando los principios de la programación orientada a objetos. De esta manera, se contempla que las rutinas de inteligencia y los comportamientos irán evolucionando conforme avance el proyecto, y siempre será fácil anexar los nuevos algoritmos desarrollados. Asimismo, si se llega a necesitar otra clase para otro tipo de robot (por ejemplo: medio, delantero, defensa derecho, defensa izquierdo), se pueden programar nuevas subclases de la clase robot en las que se implemente el método de inteligencia para este tipo de robot, pero no será necesario desarrollar todos los métodos que este robot tenga en común con los demás (por ejemplo: métodos de conexión, consulta o cambio de atributos en común, campos potenciales).

6.3. Perspectivas a futuro

6.3.1. Sistema de visión

El sistema de visión podría ser mejorado, en una siguiente versión, utilizando segmentación bayesiana o filtros de Kalman.

La segmentación bayesiana, se podría implementar de manera que además de la información de los tres canales para cada pixel, se utilice la información que corresponde al instante de tiempo anterior. De esta manera se podría tener mucha más certeza al afirmar que un cierto pixel corresponde a una cierta clase.

Los filtros de Kalman, podrían utilizarse para predecir las posiciones de los robots y pelota en instantes futuros, de manera que se pueda evitar realizar la segmentación en todos los pixeles de le imagen; se busca que la segmentación sólo se lleve a cabo en las regiones de la imagen donde sea probable que se encuentren los robots y la pelota. Esto disminuiría significativamente la carga de trabajo para el sistema de visión.

El sistema de visión fue desarrollado en Visual C++, trabajando con el sistema operativo Windows XP. Se utilizaron librerías de DirectX para capturar el buffer de video de la cámara digital conectada al puerto USB o Firewire. Estas librerías resultan muy complejas y es necesario incluir una gran cantidad de código de funciones de la librería que pueden afectar el desempeño del programa. Se pretende que en el futuro se haga una nueva versión del sistema de visión bajo el sistema operativo Linux, que utilice las librerías de captura de video OpenCV. Estas librerías son más amigables y el código resultante de un programa que las utilice no tiene la complejidad de uno que esté basado en DirectX. Además, el sistema operativo Linux es gratuito y, en muchas ocasiones, más estable y robusto que Windows.

En las nuevas reglas de la *RoboCup Small Size* se indica que la cancha debe tener una medida de $5m \ x \ 4m$, lo cual hace ya imposible el uso de webcams y hace necesario el uso de dos cámaras digitales con lentes gran angular. El uso de lentes gran angular trae consigo el problema de la distorsión geométrica. La siguiente versión del sistema de planeación/navegación deberá contemplar este problema y se tendrá que incluir una rutina que se encargue de realizar la corrección geométrica.

6.3.2. Sistema de planeación/navegación

El sistema de planeación/navegación, en un futuro, deberá contar con muchas más rutinas que contemplen más situaciones y de mayor complejidad, que se puedan dar en la cancha. Para esto, se propone que se desarrolle un sistema experto, que será el nuevo planeador. El programa actual fue diseñado de manera que la adición de un sistema experto a la rutina de inteligencia sea un asunto sencillo. El sistema experto, entonces, recibirá los datos enviados por el sistema de visión y evaluará, para cada robot, la situación en la que se encuentra, para decidir la siguiente acción que deberá llevar a cabo. Un sistema experto tendría la capacidad de almacenar muchos más comportamientos para muchas situaciones de juego muy complejas.

Los robots actualmente utilizan comandos de movimiento que reciben un ángulo de giro y una distancia de avance. Observando el desarrollo de otros equipos del mundo, se ve que este tipo de comandos son obsoletos. El robot gasta mucho tiempo en detenerse después del giro para luego comenzar con el avance y después volver a detenerse para ejecutar el siguiente comando. Es imperativo que se desarrollen las rutinas de control necesarias para que el robot pueda realizar un movimiento contínuo. El sistema de planeación/navegación está diseñado de manera que, si se realiza este cambio en el control, las modificaciones al sistema de planeación/navegación sean mínimas.

Apéndice. Manual de usuario

- *PocketPC's*:

  1. Encender la *PocketPC*.
  2. Configurar la conexión de red inalámbrica (WiFi) con los parámetros establecidos en el *Access Point* y asignar una dirección IP local.
  3. Revisar que el programa *UDPClient* se encuentre instalado en el dispositivo.
  4. En caso de no estar instalado, es necesario sincronizar el dispositivo con la PC y copiar la aplicación.
  5. Ejecutar el programa *UDPClient*.
  6. Configurar el puerto de recepción dependiendo del número de robot que se va a utilizar:
     - Robot 1: Puerto 8001
     - Robot 2: Puerto 8003
     - Robot 3: Puerto 8005
     - Robot 4: Puerto 8007
     - Robot 5: Puerto 8009
  7. Pulsar el botón "INICIAR".

- Robots:

  1. Revisar la carga de las baterías de los robots.
  2. Conectar la *PocketPC* al robot.
  3. Colocar los parches de colores al robot, de acuerdo al número de puerto que se le haya asignado a la *PocketPC* y al código de colores que le corresponde a dicho robot (tabla 3.1).

4. Colocar el parche central de color amarillo o azul, según el color que corresponda al equipo.
5. Encender el robot.

- Sistema de visión:

1. Instalar dos cámaras digitales de video sobre la cancha y conectarlas a la PC que ejecutará el sistema de visión.
2. Ejecutar dos instancias de la aplicación del sistema de visión. En cada una de ellas, se ejecutarán los siguientes pasos de la misma forma.
3. Pulsar el botón "INICIO" para que se muestre el video capturado y listo para procesarse.
4. Colocar la pelota y los robots sobre la cancha o varios parches de los diferentes colores, para hacer la calibración.
5. Calibrar las medidas de la cancha:
   - Pulsar el botón "CALIBRAR" de la sección de "Cancha".
   - En la pantalla donde se mostrará el video procesado, seleccionar la esquina inferior izquierda de la cancha y arrastrar el cursor hasta el la esquina superior derecha.
   - Escribir las coordenadas reales de cada esquina de la cancha (en metros) en los cuadros de texto de la sección "Cancha".
   - Pulsar el botón "MODIFICAR" de la sección cancha y después el botón "TERMINAR".
6. Si existe un archivo de configuración previo será necesario cargarlo en este paso.
7. De lo contrario, se hará la calibración de los colores:
   - Pulsar el botón "CALIBRAR" de la sección de "Segmentación". Esta acción toma una foto fija de la cancha y permite calibrar.
   - Elegir, uno por uno, el color que se quiere calibrar y dar click tres veces sobre la imagen fija de la cancha, sobre los parches del color en cuestión o sobre la pelota.
   - Al haber dado el tercer click, el programa emite un sonido que indica que el color ha sido calibrado.
   - Pulsar el botón "TERMINAR" de la sección de "Segmentación"
8. Pulsar el botón "PROCESAR" para que se muestre el video procesado.

9. En la sección de "Formación de BLOBS" se puede aumentar el área mínima que deben tener las regiones de color encontradas para que sean consideradas como parches. De esta manera se puede eliminar ruido.
10. En la sección de "DISTANCIA" se puede aumentar el radio en el que se deben encontrar los parches de colores alrededor del parche central (azul o amarillo) para que sean tomados en cuenta como parte del robot.
11. Si es necesario ajustar la calibración de alguno de los colores, es necesario seleccionar el color, pulsar el botón "VER" y luego mover las barras de desplazamiento para la calibración.
12. Introducir en la sección de "Comunicaciones" la dirección IP del equipo que se encargará de correr el Director Técnico.
13. Elegir el *frame rate* que se desea utilizar en en la sección de "Comunicaciones".
14. Pulsar el botón "ABRIR PUERTO" en la sección de "Comunicaciones".

- Sistema de planeación/navegación:

1. Ejecutar la aplicación del Director Técnico o sistema de planeación/navegación.
2. Ingresar las dimensiones de la cancha en los cuadros de texto de la sección "Dimensiones de la cancha".
3. Establecer la dirección hacia donde tiran los robots en la sección "Dirección de juego.
4. Si es necesario, se puede modificar la distancia de la portería a la que se debe situar el portero (también se puede cambiar cuando ya hayan empezado a jugar).
5. Pulsar el botón "ESCUCHAR" servidor 1.
6. Si se están ejecutando dos instancias del sistema de visión, es necesario pulsar también "ESCUCHAR" servidor 2.
7. Pulsar el botón "JUEGUEN".
8. Ingresar las direcciones IP de las *PocketPC* que se vayan a utilizar, tomando en cuenta el número de robot al que pertenecen.
9. Pulsar los botones "CONECTAR" para activar cada una de las conexiones. En este momento, los robots empezarán a jugar.

10. Durante el juego se pueden modificar las constantes de los campos potenciales, la distancia de avance del robot en cada movimiento (paso), el ángulo de dirección de disparo (arco), así como una constante de corrección para el ángulo de giro.

Bibliografía

[1] Ballard, D. H., and Brown, C. M. *Computer Vision.* Ed. Prentice Hall, Primera edición, E.U., 1982.

[2] Davies, E. *Machine Vision: Theory, Algorithms, Practicalities.* Ed. Academic Press, Primera edición, E.U., 1990.

[3] García, M., and Manuel, F. *UNIX. Programación avanzada.* Ed. RA-MA, Segunda edición, España, 1996.

[4] Gonzalez, R. C., and Woods, R. E. *Digital Image Processing.* Ed. Prentice Hall, Segunda edición, E.U., 2002.

[5] Jones, J. L. *Robot Programming: A practical Guide to Behavior-Based Robotics.* Ed. McGraw-Hill, E.U., 2004.

[6] Kim, J. A framework for roadmap-based navigation and sector-based localization of mobile robots. *(Tesis) Texas A&M University* (2004).

[7] Kruse, R. L., Leung, B. P., and Tondo, C. L. *Data structures and program design in C.* Prentice Hall, 2da. Edición, E.U., 1996.

[8] Leon-García, A. *Probability and Random Processes for Electrical Engeniee-ring.* Ed.Addison-Wesley, Segunda edición, E.U., 1994.

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

[10] Murphy, R. R. *Introduction to AI Robotics.* Ed. Massachusetts Institute of Technology, E.U., 2000.

[11] Pajares, G., and de la Cruz, J. M. *Visión por computador: Imágenes digitales y aplicaciones.* Alfaomega Ra-Ma, México, 2002.

[12] Ribeiro, M. I. Obstacle avoidance. *The Robotics WEBook* (2005).

[13] Roberts, D. *Developing for the Internet with Winsock*. Ed. Coriolis Group Books, E.U., 1995.

[14] Rodríguez G., F. J. Sistema de reconocimiento de patrones usando procesamiento de imágenes para localización de robots móviles. *(Tesis) Facultad de Ingeniería, UNAM* (2007).

[15] Savage, J., Billinghurst, M., and Holden, A. Virbot: A virtual reality robot driven with multimodal commands. *en Proceedings de ECAI-98: The 13th European Conference on Artificial Intelligence* (1998).

[16] Savage, J., Márquez, E., Pettersson, J., Trygg, N., Petersson, A., and Wahde, M. Optimization of waypoint-guided potential field navigation using evolutionary algorithms. *en Proceedings de la International Conference on Intelligent Robots and Systems 2004 IEEE/RSJ (IROS 2004, Japon)* (2004).

[17] Savage, J., and Schaarschmidt, W. Comparison of two path planning methods for mobile robots. *en Proceedings del Congreso Nacional de Control Automático (AMCA 2003, Ensenada, Baja California)* (2003).

[18] Sohaib, K., and Mubarak, S. Object based segmentation of video using color, motion and spatial information. *IEEE Int'l Conference on Computer Vision and Pattern Recognition* (2001).

[19] Umbaugh, S. E. *Computer Vision and Image Processing: A practical Approach Using CVIPtools*. Ed. Prentice Hall, Primera edición, E.U., 1998.

[20] URL. Robocup.
http://www.robocup.org.

[21] URL. Robocup small size robot league.
http://www.cs.cmu.edu/ brettb/robocup.

[22] URL. Small size robot league rules.
http://small-size.informatik.uni-bremen.de/rules:main.

[23] URL. Wikipedia: Blob.
http://es.wikipedia.org/wiki/BLOB.

[24] URL. Wikipedia: Linked list.
http://en.wikipedia.org/wiki/Linked_list.

[25] URL. Wikipedia: Robot. http://en.wikipedia.org/wiki/Robot.

[26] URL. Wikipedia: Run-length encoding.
http://en.wikipedia.org/wiki/Run-length_encoding.