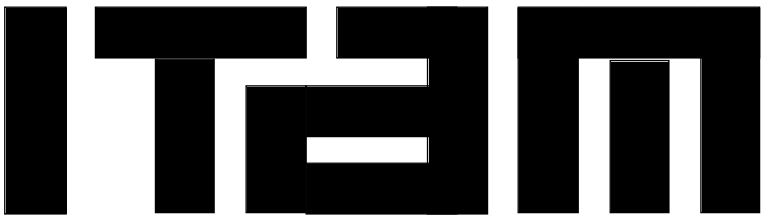
INSTITUTO TECNOLÓGICO AUTÓNOMO DE MÉXICO




# SISTEMA DE INTELIGENCIA ARTIFICIAL PARA EL CONTROL DE ROBOTS AUTÓNOMOS "SMALL SIZE"

T E S I S

QUE PARA OBTENER EL TÍTULO DE

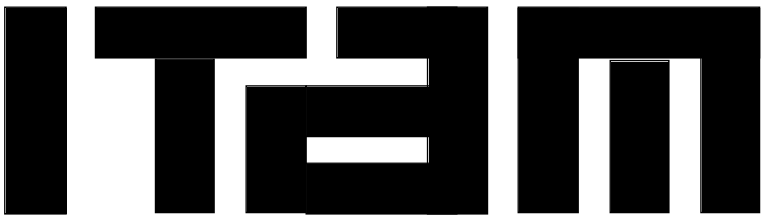
**INGENIERO EN COMPUTACIÓN**

P R E S E N T A

**ERNESTO TORRES VIDAL**

MÉXICO, D.F. ABRIL 2009

INSTITUTO TECNOLÓGICO AUTÓNOMO DE MÉXICO




# SISTEMA DE INTELIGENCIA ARTIFICIAL PARA EL CONTROL DE ROBOTS AUTÓNOMOS "SMALL SIZE"

T E S I S

QUE PARA OBTENER EL TÍTULO DE

**INGENIERO EN TELEMÁTICA**

P R E S E N T A

**ERNESTO TORRES VIDAL**

MÉXICO, D.F. ABRIL 2009

# ÍNDICE GENERAL

# Índice de figuras

# ÍNDICE DE TABLAS

# Capítulo 1

## INTRODUCCIÓN

El presente trabajo aborda el diseño y la implementación del sistema de Inteligencia Artificial (IA) utilizado por el equipo *Eagle Knigths* (EK) del Laboratorio de Robótica del ITAM dentro de la liga *Small Size* (SSL) de *RoboCup*.

El primer capítulo introduce al proyecto de *RoboCup*, describe la arquitectura general de los equipos que participan en esta liga y define el objetivo, la justificación, los trabajos relacionados con el proyecto (trabajos de tesis, un artículo publicado y la documentación generada para el curso de Robótica), así como el alcance y la organización del documento.

### 1.1. RoboCup

*RoboCup* es un proyecto internacional para promover la investigación y educación sobre inteligencia artificial a través de competencias integradas por robots autónomos [26]. El objetivo oficial del proyecto es conseguir que un equipo de robots humanoides sea capaz de vencer al campeón de la Copa Mundial de la FIFA para el año 2050.

El Laboratorio de Robótica del ITAM es parte activa de la iniciativa *RoboCup* dentro de las ligas *Standard Plaftorm* (SPL) y *Small Size* con el equipo EK [22]. Dicho laboratorio, integrado por alumnos de las Ingenierías del ITAM [19], promueve el desarrollo de tecnología en las áreas de robótica, inteligencia artificial, visión por computadora y comportamientos autónomos.

El equipo EK ha participado en la liga SSL desde el año 2003, obteniendo el tercer lugar en el *RoboCup USOpen* y el segundo lugar en el mismo torneo en el año 2004. En 2005 obtuvo el campeonato en el *RoboCup Latin American Open* en Sao Luis, Brasil. El equipo ha participado también en las siguientes competencias mundiales de *RoboCup*: Osaka, Japón 2005 [23]; Bremen, Alemania 2006 [24]; Atlanta, Estados Unidos 2007 [25] y Suzhou, China 2008 [8], destacando la obtención del séptimo lugar en la competencia de Atlanta.

El presente trabajo se enmarca en la liga *Small Size* de la competencia *RoboCup soccer* y consiste en el diseño e implementación del sistema de IA del equipo *Eagle Knights*.

### 1.2. Arquitectura completa de un equipo de la liga SSL

De acuerdo con las reglas establecidas por *RoboCup* [27] para la liga SSL, dos equipos de cinco robots cada uno juegan fútbol en una cancha de 7.4 m de largo x 5.4 m de ancho con una pelota de golf color naranja. Las dimensiones de los robots no deben ser mayores a las de un cilindro de 180 mm de diámetro y 150 mm de altura.

En general, la arquitectura de un equipo de esta liga está integrada por tres componentes principales: los robots, el sistema de visión [12] y el sistema de IA. Se cuenta también con el *referee box* (árbitro), sistema externo controlado por un árbitro auxiliar, que envía comandos a los dos equipos participantes durante el partido para indicar el inicio y el fin de los tiempos, la marcación de los goles, las notificaciones y las infracciones cometidas durante el mismo [31].

Los componentes del equipo EK desarrollados dentro del Laboratorio de Robótica son: el sistema de visión, el sistema de IA, así como el *hardware* y *software* de los robots; mientras que el *referee box* es un *software* estándar desarrollado por la SSL. La interacción y el flujo de información entre estos componentes se pueden visualizar en la figura 1.1.

Cada robot (figura 1.2) cuenta con actuadores, sensores internos y un sensor externo. Los actuadores son cuatro motores en una configuración omnidireccional [30] y un motor para controlar la pelota por medio de una barra central [15], así como un sistema de pateo de la pelota. Los sensores internos son: un led infrarrojo y un fototransistor para detectar si la pelota está cerca para patearla, así como las señales que genera cada motor para conocer su velocidad y el sentido de su movimiento. El sensor externo es un sistema de visión global que permite la localización de los robots y de la pelota en el campo de juego; utiliza dos cámaras de video colocadas a 4 m de altura del campo de juego, patrones

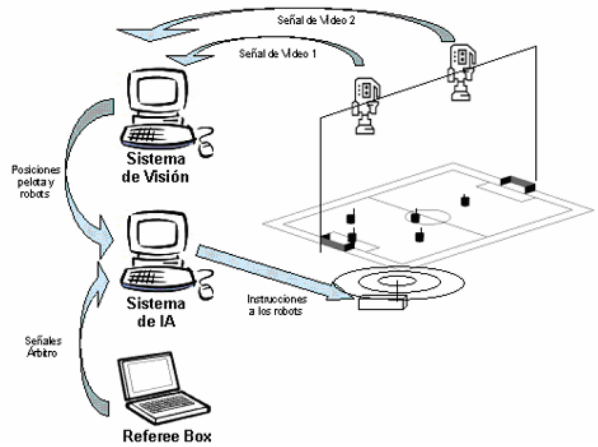



***Figura 1.1.*** *Arquitectura de un equipo de la liga SSL.*

codificados de colores en la parte superior de los robots y una computadora para el procesamiento del video.

Para la identificación de los robots es necesario utilizar un parche central que debe ser de color amarillo o azul y un número opcional de parches auxiliares de diferentes colores (figuras 1.2 y 1.3).

Los sistemas de visión y de IA se implementan en computadoras externas, lo cual presenta ciertas ventajas frente a las arquitecturas en las que estos sistemas se integran a los robots, ya que las restricciones de procesamiento inherentes a robots con capacidades limitadas se eliminan, la visión global permite conocer con mayor certeza la posición de los robots y de la pelota dentro de la cancha y, por último, la toma de decisiones de los cinco robots se centraliza en un solo sistema. Estas ventajas permiten que los robots puedan ser controlados de forma autónoma a velocidades superiores a 2 m/s y que los partidos se caractericen por tener un alto grado de colaboración en equipo.

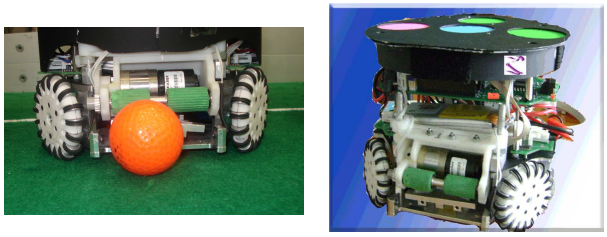


***Figura 1.2.*** *Robot EK de la liga SSL.*

***Figura 1.3.*** *Codificación de parches del equipo de robots SSL.*

El sistema de IA recibe la información del sistema de visión y decide la acción que debe realizar cada robot con base en esta información y en el estado de juego definido por el *referee box*. Una vez seleccionada la acción es necesario definir la ruta que el robot debe seguir en el campo de juego para evadir obstáculos. Finalmente, determina el vector de movimiento de cada uno de los robots, tomando en cuenta un modelo de control que permite al robot realizar la acción deseada de manera correcta. Estos vectores de movimiento, junto con las instrucciones sobre el uso de sus dispositivos (sistemas de pateo y *dribbler*) son enviadas a cada uno de los robots por medio de un transmisor inalámbrico. Es importante señalar que cuando se requiere controlar a los robots de forma autónoma, el correcto funcionamiento del sistema de IA depende en su totalidad del sistema de visión que es la principal fuente de información.

### 1.3. Objetivo

El objetivo de este trabajo es el diseño e implementación del Sistema de Inteligencia Artificial que se desarrolló para ser utilizado por el equipo *Eagle Knights* en la liga SSL de *RoboCup* en las competencias de los años 2005, 2006, 2007 y 2008.

### 1.4. Justificación

El presente trabajo describe el desarrollo del Sistema de Inteligencia Artificial que ha ido evolucionando e integrando nuevos algoritmos y que ha sido utilizado desde el año 2005 en más de 20 partidos durante las competencias de *RoboCup*. Además, este sistema ha sido utilizado en el curso de Robótica [21] para el control de otro tipo de robots como los LEGO NXT, implementando una interfaz *Bluetooth* de salida y en el curso de Sistemas de Comercio Electrónico [20] para controlar robots Lego NXT por medio de la recepción de comandos vía teléfono celular mediante servicios *web*.

El sistema de IA, además de contribuir con los objetivos del *RoboCup* y de haber sido utilizado como material de apoyo en los cursos mencionados, constituye una plataforma que integra conocimientos de diversas áreas del plan de estudios de las carreras de Ingeniería en Computación e Ingeniería en Telemática como la inteligencia artificial, los modelos de simulación, las gráficas por computadora, los algoritmos de planeación, control y navegación, los sistemas distribuidos y las comunicaciones inalámbricas. Estas áreas son pilares fundamentales en la investigación de la robótica que, además de aplicarse a partidos de futbol, se utilizan para aplicaciones como el rescate de víctimas en zonas de desastres químicos o sísmicos, el ensamblaje automotriz, la medicina asistida por computadora, las exploraciones espacial y submarina, así como la automatización de medios de transporte.

### 1.5. Trabajos relacionados

Existen diversos trabajos que describen a detalle otros módulos de los sistemas que conforman la arquitectura completa del equipo EK.

En el año 2004, Luis Martínez Gómez describió en su tesis titulada *"Sistema de Visión para el Equipo de Robots Autónomos del ITAM"* [12], el procesamiento de imágenes para

la localización de los robots y la pelota con base en la identificación de regiones de colores en tiempo real.

Como parte del desarrollo de esta tesis se han realizado nuevos trabajos de investigación del sistema de visión para la integración de nuevas soluciones como el publicado en el *Latin American Robotics Symposium* (LARS) en el año 2008 [10]. Este trabajo explica una metodología basada en redes neuronales para el aprendizaje de la segmentación de los colores de una imagen en tiempo real. También se documentó la versión actual del sistema de visión que puede ser consultada en la página del curso de Robótica [29].

El diseño y construcción de los robots omnidireccionales utilizados para las competencias de los años 2004 y 2005 son descritos por David Sotelo en su tesis titulada *"Diseño e Implementación de los Robots F180 del ITAM"* [14].

La arquitectura del sistema presentada en este trabajo tiene como precedente la desarrollada en la tesis de Francisco Moneo Soler que se titula *"Sistema de Planeación de Alto Nivel para Robocup"* [13], utilizada en las competencias de los años 2003, 2004 y 2005 en donde se describe a detalle la primera versión del sistema de IA del equipo EK, enfocada especialmente a la utilización de máquinas de estados en la estrategia de juego y los campos potenciales para la evasión de obstáculos.

### 1.6. Alcance

El alcance de esta tesis es el diseño e implementación del Sistema de Inteligencia Artificial que tiene como finalidad que el equipo de robots EK juegue futbol de forma autónoma, es decir, sin interacción humana durante su ejecución en los partidos, interactuar fácilmente con el usuario y permitir realizar pruebas para validar el correcto funcionamiento de los robots y el sistema de visión.

El diseño de la solución abarca la descripción de la arquitectura del sistema y sus módulos.

El alcance del sistema tiene las siguientes funcionalidades:

- **Sistema de visión y referee box:** Proveer las interfaces necesarias para recibir la información enviada por el sistema de visión y los comandos utilizados para controlar los estados de juego durante los partidos.
- **Visualización y control de pruebas:** Proveer un ambiente gráfico en el cual se pueda visualizar el funcionamiento del sistema, integrar el uso de un controlador externo y diseñar una interfaz de control que permitan probar el funcionamiento de los robots.

- **Control de movimientos:** Desarrollar un modelo de control omnidireccional que permita al robot seguir las trayectorias requeridas, integrando las restricciones del sistema de visión y el envío de la información por medio de un transmisor inalámbrico, así como un modelo de aceleración y desaceleración.
- **Simulador de movimientos:** Diseñar un simulador dinámico basado en el modelo omnidireccional que permita programar y validar las acciones que deben realizar los robots
- **Detección del movimiento de la pelota:** Obtener el vector de movimiento de la pelota a partir de las mediciones del sistema de visión y que pueda ser utilizada en los comportamientos de los robots.
- **Estrategia de juego:** Desarrollar un módulo que permita la implementación de varias estrategias de juego, coordinar a los robots y realizar tareas en equipo utilizando un modelo basado en conocimiento y realizar las acciones definidas por la estrategia de juego a partir de la definición de la coordenada y orientación final y las velocidades de movimiento del robot.
- **Control de trayectorias:** Solucionar el problema de seguir una trayectoria a partir de una serie de puntos de forma navegable por el robot y diseñar un algoritmo que permita generar trayectorias con la finalidad de evadir obstáculos y encontrar una ruta adecuada.

### 1.7. Organización del documento

El trabajo de esta tesis se distribuye en capítulos de la siguiente manera: El primero explica el contexto y planteamiento del problema, los objetivos y el alcance; el segundo, el marco teórico requerido para el análisis y la solución; el tercero resume los requerimientos del sistema; el cuarto describe el diseño de la arquitectura, los módulos del sistema y los algoritmos utilizados para resolver los problemas planteados; el quinto, la implementación del sistema de IA, enfatizando la interacción con los demás sistemas y con el usuario; el sexto trata acerca de las pruebas realizadas al desarrollar los algoritmos del sistema y los resultados obtenidos después de probarlo junto con el sistema de visión y los robots; el séptimo describe las conclusiones y las líneas futuras del trabajo a realizar en el Laboratorio de Robótica con base en el presente trabajo. En el Anexo se describe una guía de instalación del sistema, los comandos y notificaciones del *referee*, así como un glosario de términos.

# Capítulo 2

## Marco Teórico

Este capítulo introduce al tema de la aplicación de la inteligencia artificial a la robótica y analiza los enfoques de la IA orientados a la IA simbólica y al proceso de representación de la lógica por medio de un modelo basado en conocimiento como son los métodos de búsquedas heurísticas y los sistemas expertos. También integra el esquema de percepción, cognición y acción necesarios para realizar el modelo de un robot autónomo.

Posteriormente se introduce al tema de la navegación en donde se explica la teoría de la generación de trayectorias a partir de una serie de puntos, los campos potenciales y la planeación de rutas por medio de árboles de exploración aleatoria.

Finalmente, se aborda la teoría del filtro de Kalman utilizado para obtener el vector de movimiento de la pelota a partir de las mediciones del sistema de visión y la teoría de controladores PID utilizados para el control de movimientos de los robots .

### 2.1. Robótica

La robótica es la técnica que aplica la informática al diseño y construcción de aparatos que realizan operaciones o trabajos orientados comúnmente a aplicaciones industriales y de navegación.. La investigación en robótica ha seguido dos enfoques: El tradicional que

utiliza a la inteligencia artificial simbólica basada en conocimientos y, el más reciente, que toma inspiración de la naturaleza y está basado en comportamientos [2].

## 2.2. Inteligencia artificial

La inteligencia artificial estudia cómo lograr que las máquinas realicen tareas que por el momento, son mejor realizadas por los seres humanos. Tiene su origen en la computación, la ingeniería y las matemáticas. Su principal objetivo es la construcción de sistemas que demuestren comportamiento inteligente y realicen tareas complejas con un nivel de competencia igual o superior al mostrado por un ser humano experto [1].

Los tipos de problemas que la inteligencia artificial busca resolver, frecuentemente mediante el uso de la heurística, son complicados porque no se conoce una solución algorítmica o se conoce el algoritmo, pero tomaría mucho tiempo encontrar la solución.

Dentro de las áreas de estudio de la inteligencia artificial se encuentran: el aprendizaje, el reconocimiento de patrones, la visión por computadora, el entendimiento del lenguaje natural y la robótica.

Existen muchos enfoques de la inteligencia artificial, de los cuales dos son de interés: la inteligencia artificial simbólica y la inteligencia artificial por conectividad. El primero utiliza la lógica para representar el conocimiento y, de esta manera, realizar un proceso deductivo; por ejemplo, un sistema experto convierte la información sobre algún problema acotado en conocimiento que se representa y es manipulable en una computadora. El segundo radica en el mecanismo computacional de las redes neuronales, las cuales se componen de pequeñas unidades llamadas neuronas interconectadas en forma de red. Las neuronas son unidades independientes que desempeñan cálculos locales. Mediante la cooperación y el intercambio de información local son capaces de aprender una tendencia con base en un conjunto discreto de datos [1]; por ejemplo, para clasificar los valores de los pixeles de una imagen en un conjunto discreto de clases [10].

### 2.2.1. Modelo basado en conocimiento

Para lograr la autonomía de los robots en un modelo basado en conocimiento se lleva a cabo la descomposición de los procesos que el robot debe realizar en tareas independientes que posteriormente se unirán [2]. La primera es la interpretación de los datos procedentes

de los sensores. La segunda es el modelado del entorno en el que va a operar el robot. La tercera es la planificación de un conjunto de acciones que el robot realizará para la consecución de un comportamiento determinado. La cuarta y última consiste en la ejecución de las acciones por medio de actuadores que logran los movimientos del robot como se muestra en la figura 2.1.

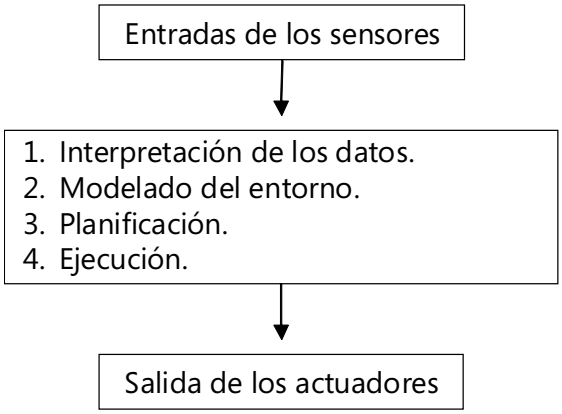



***Figura 2.1.*** *Enfoque de la robótica en un modelo basado en conocimiento.*

Existen algunas variaciones dentro de esta arquitectura relacionadas principalmente con la cantidad de información que se tiene del entorno:

a) Cuando se tiene un conocimiento total del entorno donde el robot va a desempeñar sus tareas, éste se puede modelar y definir una base de conocimiento (como sucede en la SSL). En caso de que este entorno sufra alguna modificación, es necesario actualizar el modelo y rediseñar la base del conocimiento.

b) Cuando el entorno es parcialmente conocido es posible realizar un preprocesamiento del mismo que permita diseñar un modelo que describa con precisión este entorno y de esta forma definir una base de conocimiento adaptable (como sucede en la Liga *RoboCup Home*).

c) Cuando no se tiene conocimiento alguno del entorno, ya sea porque cambia dinámicamente o porque no se pueden definir con exactitud sus variaciones mediante un preprocesamiento, es necesario recurrir a soluciones más complejas tanto en la percepción como en la actualización de la base del conocimiento. Este es el principal reto de la robótica en la actualidad.

Roodney Brooks [5] propone un modelo basado en comportamientos que cambia completamente el enfoque de la autonomía de los robots tradicionalmente basado en el conocimiento. Esta propuesta surge por la dificultad que se ha presentado en la robótica en la realización de un proceso cognitivo para modelar el entorno. En este modelo alterno, los robots realizan una acción hasta que por medio de sus sensores, detectan alguna condición de cambio que los hace modificar su comportamiento siguiendo una máquina de estados. Las aplicaciones basadas en este enfoque parten de comportamientos simples que se van sofisticando construyendo comportamientos más complejos.

### 2.2.2. Representación del conocimiento

Dentro de la inteligencia artificial simbólica se encuentran diversas técnicas de razonamiento. Por lo general, estas técnicas son adecuadas para problemas en los que el modelo del mundo es completo, consistente e inalterable. Algunas de las técnicas utilizadas para la solución de estos problemas son: la búsqueda a lo ancho y la búsqueda en profundidad. También existen soluciones heurísticas como la búsqueda por escalada, la búsqueda del primero mejor (*best-first-search*) y el algoritmo A* [1].

Cuando se intenta resolver el problema de lograr que un equipo de robots juegue futbol, es necesario recurrir a una metodología que en lugar de estar basada en un algoritmo de búsqueda general nos ayude a representar y manipular el conocimiento para encontrar una solución adecuada.

### 2.2.3. Propiedades de un sistema basado en conocimiento

Un buen sistema de representación del conocimiento debe cumplir con las siguientes propiedades [1]:

- **Suficiencia de la representación:** capacidad de representar todos los tipos de conocimiento necesarios en el dominio.
- **Suficiencia deductiva:** capacidad para manipular las estructuras de la representación con el fin de obtener nuevas estructuras que correspondan con un nuevo conocimiento deducido a partir del anterior.
- **Eficiencia deductiva:** capacidad de incorporar información adicional en las estructuras de conocimiento con el fin de que los mecanismos de inferencia puedan seguir las decisiones más prometedoras.
- **Eficiencia en la adquisición:** capacidad de adquirir nueva información con facilidad.

### 2.2.4. Búsquedas heurísticas

Una búsqueda heurística no garantiza el éxito de la misma forma que lo hace un algoritmo o una decisión procedural, pero es útil para la gran mayoría de los casos. Una forma sencilla de búsqueda heurística es *hill-climibing* que consiste en darle al programa una función de evaluación que pueda aplicarse al estado actual del problema para poder obtener un estimado de qué tan bien van ocurriendo las cosas [34].

El algoritmo básico de *hill-climbing* es:

1. A partir del estado actual se aplican reglas que generan un nuevo conjunto de posibles soluciones.
2. Si cualquier estado del nuevo conjunto es la solución, termina con éxito. De otra forma, se selecciona el mejor estado dentro de este conjunto y se regresa al paso anterior.

Un problema que se presenta con este algoritmo es cuando no hay una metodología confiable para determinar qué tan buena es una solución para el estado actual o cuando todo el conjunto de soluciones es igualmente bueno o malo y no se tiene un camino claro. Finalmente, se encuentra el problema del máximo local, en donde una evaluación inicial lleva a una posición a partir de la cual es imposible alcanzar la solución por ubicarse a partir de otra posición en el árbol.

Otra forma de búsqueda heurística es *best-first-search* que tiene mejores propiedades para solucionar el problema de tomar decisiones irrevocables basadas en información local. Otros algoritmos más complicados como A*[1] proponen una solución similar con la idea de que se debe mantener el rastro de los nodos que ya se han visitado y considerar la opción de regresar a algún punto anterior.

Los sistemas expertos [3] permiten resolver las dificultades de la búsqueda a partir de la representación explícita del conocimiento que el experto posee en algún dominio y las estrategias que utiliza para razonar acerca de lo que se conoce.

### 2.2.5. Sistema experto

Un sistema experto es un programa inteligente que utiliza el conocimiento y procedimientos de inferencia para resolver problemas que son lo suficientemente complicados como para requerir conocimiento de un humano experto para su solución [3].

El conocimiento de un sistema experto consiste en un conjunto de hechos y reglas heurísticas en lugar de utilizar un enfoque algorítmico como los programas convencionales.

Los cuatro componentes principales dentro de la arquitectura de un sistema experto son: la adquisición del conocimiento, la base del conocimiento, el motor de inferencia y la interfaz de entrada y salida.

#### 2.2.5.1. Base del conocimiento

La base del conocimiento contiene el dominio específico y el control del conocimiento que se utiliza para resolver los problemas. El conocimiento puede ser definido por los expertos o adquirido por medio de técnicas de aprendizaje de máquina a partir de un conjunto de datos. El modelo conceptual se convierte en un modelo representable en la computadora en el proceso llamado representación del conocimiento. Una forma común de hacer esto es por medio de reglas en la forma IF-THEN. Estas reglas son procesadas para resolver conflictos, determinando cuáles son susceptibles de ejecutarse y eligiendo la más apropiada.

#### 2.2.5.2. Motor de inferencia

El motor de inferencia contiene algoritmos generales que procesan el conocimiento almacenado en la base del conocimiento para resolver un problema determinado. Este motor es general por lo que no depende del dominio de la aplicación. El motor de inferencias se basa en una regla de inferencia, que es el proceso de deducir nuevo conocimiento a partir del conocimiento existente en una estrategia de búsqueda.

#### 2.2.5.3. Interfaz de entrada y salida

Esta interfaz define la forma en la que el sistema experto interactúa con el usuario u otros sistemas. Esta interacción puede ser por medio de una GUI (*Graphical User Interface*) o por medio de una librería para comunicarse con algún lenguaje de programación tradicional.

#### 2.2.5.4. Adquisición del conocimiento

Este componente permite la recolección del conocimiento necesario para resolver un problema en alguna aplicación determinada y adecuarlo al ambiente para manipulación computacional.

### 2.2.6. CLIPS

CLIPS es una herramienta que sirve para programar un sistema experto; fue desarrollada por la STB *(Software Technology Branch)* de la NASA y liberada en 1986 [4]. Esta herramienta facilita el desarrollo de *software* para modelar el conocimiento humano y fue diseñada para ser integrada a lenguajes de programación como C++ o *Java*. Además, provee los componentes principales que forman parte de la arquitectura de un sistema experto.

Existen tres formas de representar el conocimiento en CLIPS:

1. **Reglas:** conocimiento heurístico basado en experiencia.
2. **Funciones:** conocimiento procedural.
3. **Programación orientada a objetos:** conocimiento procedural.

### 2.2.7. Percepción, cognición y acción

La percepción implica la interpretación por medio de sensores de lo que sucede en el mundo real.

La acción incluye la habilidad de navegar por el mundo y manipular objetos [1]. Estos procesos se incorporan al proceso cognitivo que es estudiado por la IA (figura 2.2).

Si se realiza un análisis comparativo entre la IA y la robótica se encuentra que usualmente los programas de IA operan en un mundo simulado en una computadora, mientras que los robots operan en el mundo real. Esto hace que la problemática no sólo se concentre en las metodologías utilizadas para modelar el conocimiento, sino que también es necesario abordar los problemas de percepción, es decir, manipular la información obtenida a partir de una señal analógica para poder interpretar objetos y situaciones que para nosotros son triviales en un partido de futbol y de acción, es decir, lograr movilidad, manipulación de objetos (como la pelota) y navegación. La tabla 2.1 menciona las principales diferencias entre el mundo real y el mundo simulado.

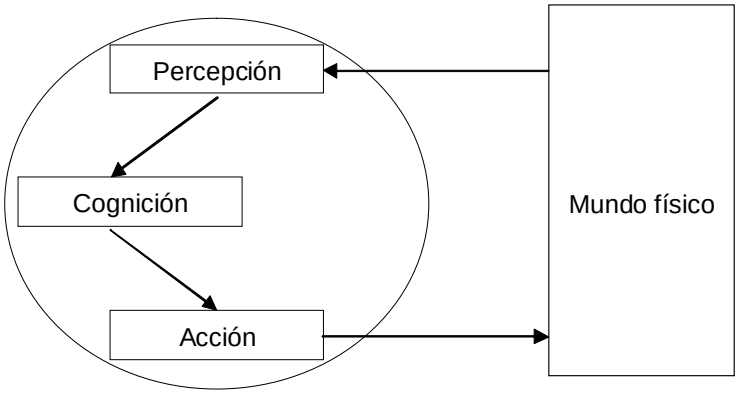



***Figura 2.2.*** *Modelo de percepción, cognición y acción para un robot autónomo.*

| Observación | Mundo simulado | Mundo real |
|---|---|---|
| *Entrada para un sistema de IA* | Simbólica en su forma | Una o varias señales analógicas |
| *Requerimientos para percibir y actuar en el mundo* | *Hardware* especializado | *Software* |
| *Incertidumbre* | No existe | Generada por la inexactitud y precisión limitada de los sensores |
| *Tiempo de procesamiento* | No hay limitaciones fuertes en el tiempo dedicado a solucionar un problema | Se debe dar una solución en tiempo real. |
| *El entorno que se modela* | No ocurren cambios inesperados, es predecible y certero. | Impredecible, dinámico e incierto. No se puede esperar una descripción correcta y completa por lo que se necesitan márgenes que permitan que el robot sea flexible ante cambios inesperados. |
| *Costo de la solución* | Bajo porque se puede llegar a un plan óptimo en donde es posible verificar estas precondiciones | Alto por el uso de recursos físicos necesarios para validar precondiciones |

***Tabla 2.1.*** *Diferencias entre el mundo simulado y el mundo real.*

## 2.3. Navegación

La navegación es el movimiento de los robots dentro del campo de juego, es decir, planificar rutas y evadir obstáculos. Dentro de los problemas de navegación se encuentran dos temas. La generación de una trayectoria en función de una serie de puntos en el campo de juego y la modificación de la ruta original para integrar la evasión de obstáculos. Existen muchas técnicas utilizadas para la evasión de obstáculos y varían dependiendo de los requerimientos del problema.

Dentro de las metodologías de navegación se describen los *splines* por ser una herramienta matemática utilizada para generar rutas suaves, los campos potenciales, por ser utilizados comúnmente para la evasión de obstáculos y los árboles de exploración aleatoria, por ser el algoritmo que se tomó como punto de partida para diseñar una solución para el problema de planificación de rutas.

### 2.3.1. Generación de una trayectoria a partir de una serie de puntos

Para lograr que un robot recorra una trayectoria a partir de la definición de un conjunto de puntos intermedios o puntos de control de forma suave, se puede recurrir a la herramienta matemática de los *splines* que es comúnmente utilizada para la manipulación de brazos robóticos industriales, para el control de movimiento de robots humanoides y en otras áreas como el diseño gráfico, así como la interpolación y suavización de datos.

Un *spline* es una función definida por una familia de polinomios estrechamente vinculados. Existen diversos tipos de interpolación como: *B-Splines*, curvas de Bézier *y splines* cúbicos. Para propósitos de este problema se recurre a los *splines* cúbicos que utilizan polinomios de grado 3 que es el menor grado que permite la presencia de puntos de inflexión, su cálculo no es complejo y evita oscilaciones presentes con polinomios de grado superior (figura 2.3).

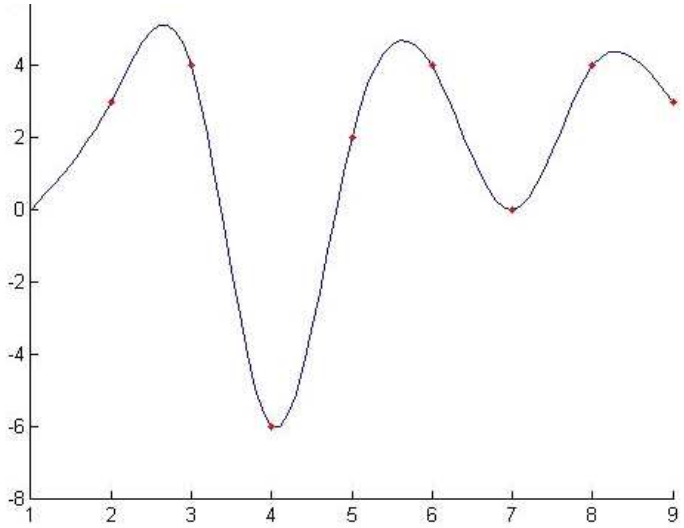



***Figura 2.3.*** *Interpolación de puntos por medio de un spline cúbico.*

### 2.3.2. Campos potenciales

Una solución al problema de la evasión de obstáculos son los campos potenciales. En este algoritmo se crea un campo ficticio a lo largo y ancho del mapa que dirige al robot desde su inicio hasta su meta, evitando todos los obstáculos que se encuentre en el camino. La meta actúa como una fuerza de atracción sobre el robot y, los obstáculos, como fuerzas repulsivas. La fuerza resultante aplicada sobre el robot puede obtenerse como la suma del total de fuerzas atractivas más la suma del total de fuerzas repulsivas [9]. La figura 2.4 muestra el resultado de aplicar una fuerza de atracción y una de repulsión sobre un punto y el vector de movimiento resultante.

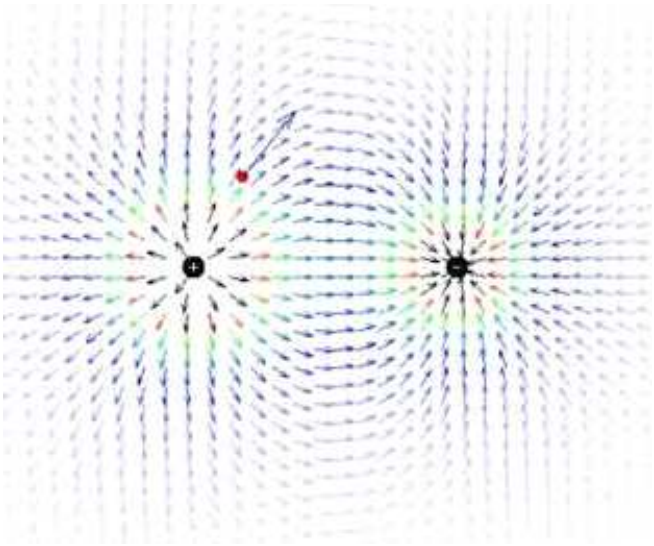


***Figura 2.4.*** *Navegación por medio de campos potenciales.*

### 2.3.3. Árbol de exploración aleatoria

Una solución para el problema de planificación de rutas es por medio de árboles de exploración aleatoria. El RRT (*Rapid-exploring Random Tree*) fue propuesto por LaValle [7] en 1998 y consiste en un árbol que explora un espacio multidimensional apoyándose en un proceso de exploración aleatoria. El algoritmo parte de la definición de un punto de inicio como raíz del árbol y una meta. La finalidad del algoritmo es encontrar un camino entre el punto de inicio y la meta en un espacio en el que deben respetarse las restricciones de movimiento impuestas por la presencia de obstáculos. El algoritmo consiste en la iteración de un proceso que tiene dos posibilidades: con probabilidad $p$, se selecciona el punto del árbol más cercano a la meta y se extiende hacia ella y con probabilidad $1 - p$, se escoge aleatoriamente un punto en el espacio y se extiende el nodo más cercano del árbol hacia este punto, siempre y cuando la extensión a realizarse no colisione con algún obstáculo. La figura 2.5 muestra el diagrama de flujo de este algoritmo.

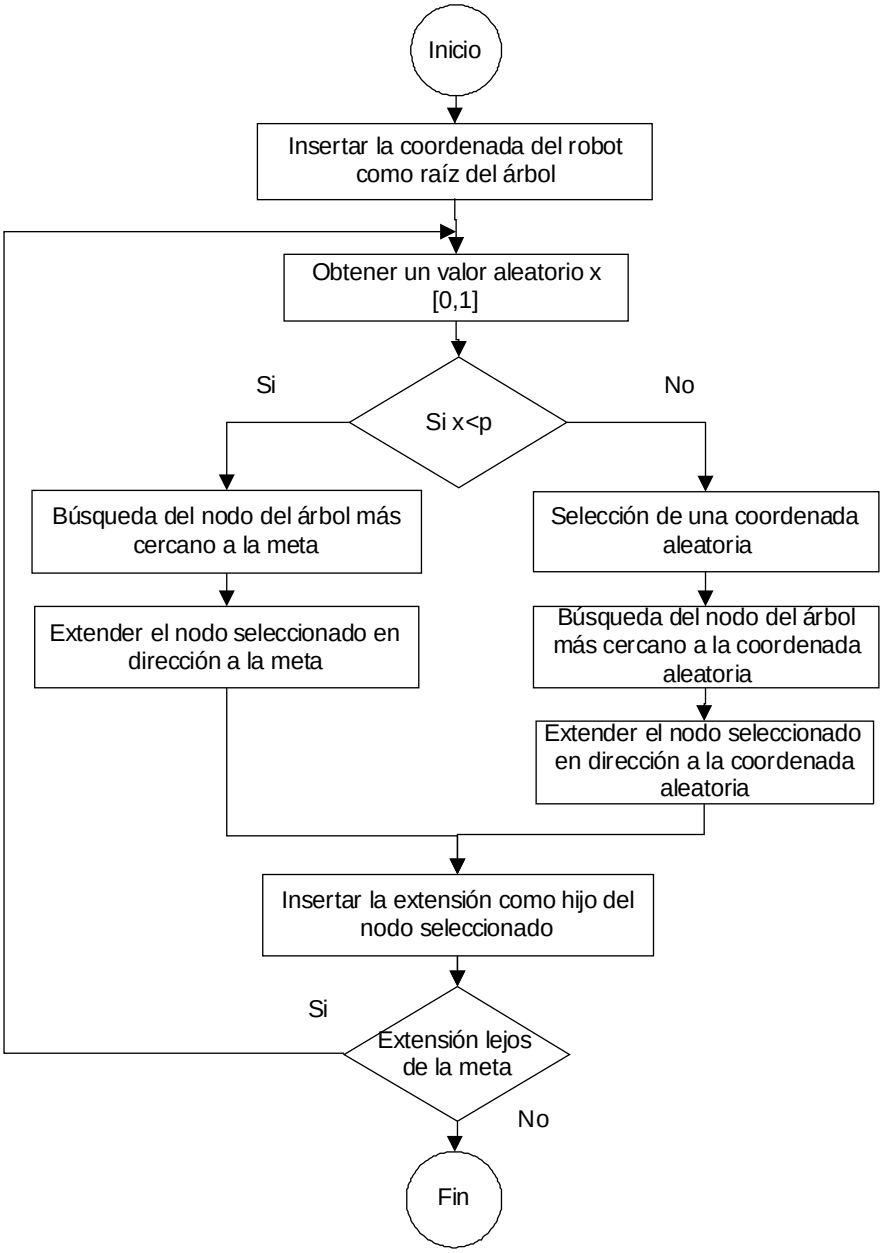



***Figura 2.5.*** *Diagrama de flujo del algoritmo RRT.*

La figura 2.6 muestra el proceso seguido por el algoritmo para una serie de seis pasos. En el *paso 1* se parte de la definición de dos puntos: el inicio y la meta. En el *paso 2*, el resultado de la evaluación aleatoria es la generación de un punto aleatorio hacia el cual se genera una extensión. En el *paso 6* se puede observar que el crecimiento del árbol en algunas iteraciones se realiza de forma aleatoria con la finalidad de explorar y en otras, dirige el nodo más cercano hacia la meta.

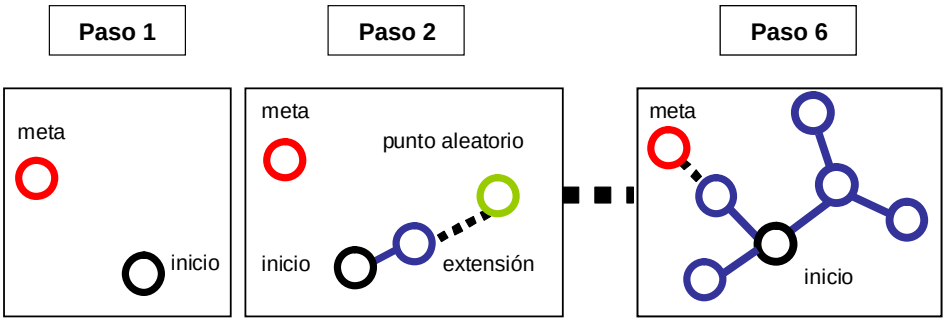



***Figura 2.6.*** *Proceso de exploración de un árbol RRT.*

## 2.4. Filtro de Kalman

El filtro de Kalman es una solución recursiva para resolver el problema de filtrado de datos lineales y discretos. El proceso, aplicado al problema del movimiento de la pelota, es discreto en el tiempo y utiliza la predicción de estados posteriores, basado en un modelo dinámico y una serie de mediciones. Tanto las mediciones como las predicciones cuentan con un cierto grado de error llamados ruido de medición y ruido del proceso (se asume ruido blanco con distribución normal).

Los procesos de medición y predicción son integrados para generar una corrección que a su vez se utiliza para generar la siguiente predicción. Las ecuaciones de predicción son las siguientes [33]:

$$XP_k = A\, X_{k-1} \tag{2.1}$$

$$PP_k = A\, P_{k-1}\, A' + Q \tag{2.2}$$

La primera ecuación muestra la transición entre los estados $X_{k-1}$, $X_k$ (vectores de dimensión $n$ x *1*) que está determinada por la matriz A ($n$ x $n$). Las matrices de covarianza del error del proceso Q($n$ x $n$) y la covarianza del error de la medición R($n$ x $n$) pueden variar en cada momento del tiempo, sin embargo, aquí se asumen constantes. La segunda ecuación permite obtener la covarianza del error de la estimación *a priori* para el siguiente estado. Las ecuaciones de corrección son las siguientes [33]:

$$XU_k = XP_k + K_k (Z_k H XP_k) \quad (2.3)$$

$$K_k = PP_k H' 1 / ((H PP_k) H' + R) \quad (2.4)$$

$$PU_k = (I - K_k H) PP_k \quad (2.5)$$

En donde $Z_k$ *(n* x *1)* es la medición. Estas ecuaciones son responsables de la retroalimentación, es decir, de incorporar una nueva medición a la estimación *a priori* para obtener una estimación *a posteriori* mejorada. A continuación se describe la notación de las variables del filtro de Kalman:

$X_{k-1}$: Vector de posición que define el estado *k-1*.
$XP_k$: Vector de posición que define la predicción en el estado *k* a partir del estado *k-1*.
$XU_k$: Vecctor que define la corrección en el estado *k*.
*A:* Matriz de transición para realizar la predicción.
*A':* Transpuesta de la matriz *A*.
*Q*: Matriz que define el ruido en el proceso de la predicción.
*R:* Matriz que define el ruido en el proceso de la medición.
$Z_k$: Vector que define la medición en el estado *k*.
$K_k$: Matriz que define la ganancia de Kalman.
*PPk:* Covarianza del error de la estimación *a priori*.
*PUk:* Covarianza del error de la estimación *aposteriori*.
*I:* Matriz identidad.
*H:* Matriz que relaciona el estado actual con la medición.

## 2.5. Controlador PID

Un controlador PID es un mecanismo de control por retroalimentación. El algoritmo para el cálculo de la salida utiliza tres factores: proporcional, integral y derivativo.

El factor proporcional determina la reacción del error actual, el integral determina la reacción basada en la suma de errores recientes y el derivativo determina la reacción basada en el cambio del error. Cada uno de estos parámetros se pondera por medio de una constante y se suma para obtener la salida del controlador. Tomando *e(t)* como la función que representa el error en el tiempo, se puede obtener la contribución de cada componente y la salida resultante:

$$P_{contribución} = K_p e(t)\,, \tag{2.6}$$

$$I_{contribución} = K_i \int_{-\infty}^{t} e(t)dt\,, \tag{2.7}$$

$$D_{contribución} = K_d \frac{de(t)}{dt}\,. \tag{2.8}$$

$$Salida = P_{contribución} + I_{contribución} + D_{contribución} \tag{2.9}$$

Para la obtención del error se toma la diferencia entre un valor de referencia que representa el valor deseado y un valor de retroalimentación (*feedback*) que representa la medición del valor real. La respuesta del controlador depende del valor de las constantes $K_p$, $K_i$, $K_d$. La figura 2.7 muestra un ejemplo de la salida de este controlador para valores $K_p = 100$, $K_i = 1$, $K_d = 1$ cuando se aplica una entrada impulso.

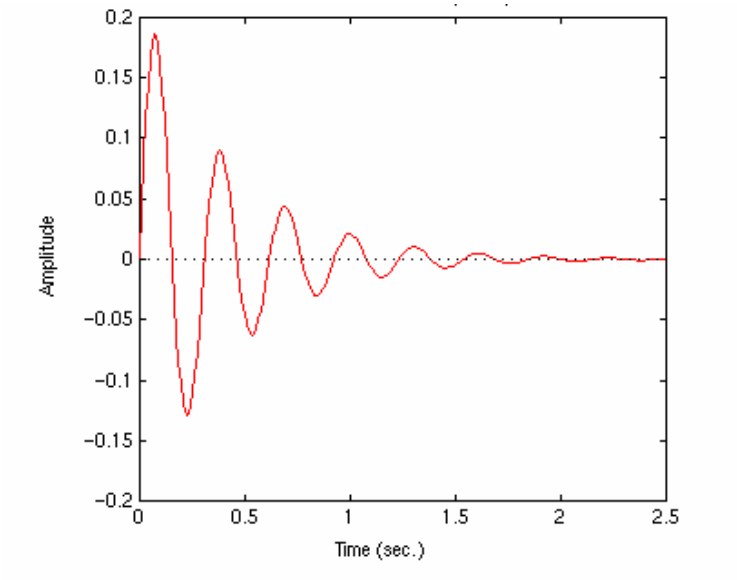



***Figura 2.7.*** *Respuesta de un controlador PID ante una entrada impulso.*

# Capítulo 3

## REQUERIMIENTOS

En este capítulo se identifican los principales requerimientos del sistema de IA, tomando como base la arquitectura utilizada en los años 2003 a 2005 [13]. El proceso de desarrollo del sistema de IA se realizó de manera cíclica, partiendo de una arquitectura bien definida y modificando los algoritmos y la implementación de cada uno de los módulos con base en las restricciones impuestas por las reglas de la liga SSL, el desempeño de los robots en los torneos, las pruebas realizadas en el laboratorio, así como en la evolución tanto del sistema de visión como del *hardware* y el *software* de los robots del 2006 al 2008.

### 3.1. Arquitectura del sistema de IA

El sistema de IA parte de la interacción con los sistemas que componen la arquitectura del sistema completo; debe contar con entradas provenientes tanto del sistema de visión como del *referee box* y con interfaces para que el usuario realice las configuraciones necesarias. También requiere de salidas para enviar la información a los robots por medio de un sistema de comunicación inalámbrico y para mostrar al usuario su funcionamiento (figura 3.1).

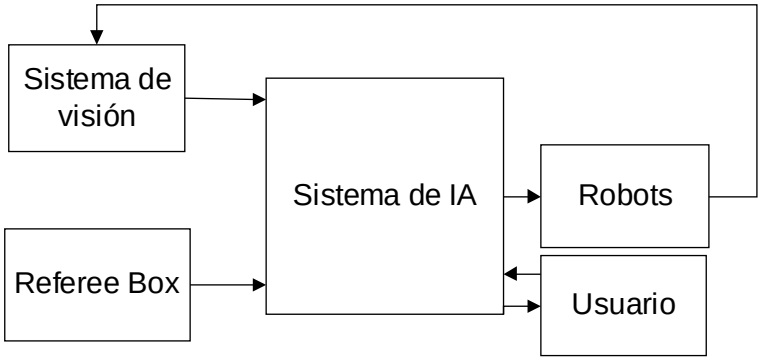



***Figura 3.1.*** *Arquitectura del sistema completo.*

### 3.2. Sistema de visión y referee box

Para que el sistema de IA interactúe con el sistema de visión es necesario contar con una interfaz de comunicación en red para recibir la localización de los robots y la pelota. Además, el sistema de IA requiere de una interfaz para recibir los comandos del *referee box* (Anexo) y generar internamente los estados de juego (figura 2.10) que puedan ser utilizados en la estrategia de juego.

### 3.3. Visualización y ambiente de pruebas

El ambiente gráfico debe permitir visualizar la posición, orientación, número, configuración de parches y uso de dispositivos de los robots para que el usuario identifique los posibles errores en la respuesta de los robots procedentes del sistema de visión, del sistema de IA o de los robots mismos. También es necesario mostrar tanto la posición como la dirección del movimiento de la pelota.

Además de esta información, el sistema debe mostrar al usuario las trayectorias que siguen los robots, así como los vectores de movimiento (translación y giro) que les son enviados.

El ambiente de pruebas debe permitir validar que la respuesta de los robots sea la correcta. Es necesario que las pruebas se realicen de tres formas:

1. Por medio de un controlador externo *(joystick)* para probar el movimiento omni-direccional del robot.

2. Utilizando el ratón de la computadora para verificar que los robots sean correctamente localizados por el sistema de visión en las diferentes regiones, seleccionando coordenadas específicas del campo de juego

3. Con una interfaz de control que permita al usuario definir los vectores de movimiento del robot.

### 3.4. Control de movimientos

El modelo de control de movimientos se implementa de forma distribuida entre el sistema de IA y los robots. Para definirlo es necesario diseñar un modelo de control omnidireccional en el robot, un protocolo de comunicación entre el sistema de IA y cinco robots, un modelo de control autónomo y un modelo de aceleración y frenado.

El modelo de control omnidireccional debe permitir controlar los motores del robot a partir de los vectores de movimiento indicados por el sistema de IA por medio de controladores PID.

Los vectores de movimiento junto con el uso de los dispositivos de los robots deben ser integrados en un protocolo de comunicación utilizando un transmisor inalámbrico compatible con el *hardware* de los robots y respetando el número de ciclos por segundo del sistema de visión.

El modelo de control autónomo genera los vectores de movimiento de los robots a partir de una coordenada y orientación determinadas, seleccionando el sentido de giro más corto y permitiendo al robot trasladarse y girar al mismo tiempo. Además, el usuario debe poder definir los parámetros necesarios para realizar este control de manera que el sistema se adapte a cambios en las dimensiones del campo de juego, la altura de las cámaras y la velocidad a la que se desea controlar a los robots.

El modelo de aceleración y frenado debe modular cambios drásticos en las velocidades enviadas a los robots para poder controlarlos a velocidades superiores a 1.5 m/seg.

### 3.5. Simulador de movimientos

El simulador de movimientos debe permitir al usuario recrear el movimiento de los robots cuando esta información no es recibida por el sistema de visión. Es necesario tomar en

consideración el modelo de control omnidireccional para simular las capacidades de translación y giro que ocurren en la realidad.

Además se requiere que el simulador permita visualizar los cambios en las velocidades de los robots y sea una herramienta para la programación de los comportamientos de los robots. El simulador debe integrar la dinámica del movimiento de la pelota para probar comportamientos que requieren mayor precisión como tirar a gol o enviar pases.

### 3.6. Detección del movimiento de la pelota

La detección del movimiento de la pelota se debe obtener de una serie de mediciones con ruido provenientes del sistema de visión. Se requiere que el modelo de predicción trabaje independientemente en cada ciclo de control por la velocidad y rapidez de los cambios de la posición de la pelota en el campo de juego. El vector generado a partir de este proceso debe ser útil para la ejecución de diversos comportamientos como la intercepción de la pelota y la recepción de pases.

### 3.7. Estrategia de juego

La estrategia de juego consiste en determinar los comportamientos en colaboración de los cinco robots durante los partidos que se realizan en función de la información proveniente del sistema de visión y de los estados de juego.

Es necesario que el modelo para definir las estrategias de juego se base en un sistema experto y que permita al usuario diseñarlas dependiendo del nivel de juego del equipo contrario.

La estrategia de juego debe responder adecuadamente a la ubicación de los robots contrarios, respetar las reglas de la liga SSL y generar evaluaciones de lo que ocurre en el campo de juego a partir de la localización de los robots y la pelota.

Los comportamientos de cada uno de los robots se definen en cada ciclo de control. La ejecución de estos comportamientos debe generar una coordenada y orientación finales para que los robots puedan ser controlados de forma similar a como lo hace el usuario por medio del ambiente de pruebas, pero de forma autónoma.

### 3.8. Control de trayectorias

El problema de la navegación se puede resolver por medio de metodologías de planificación de trayectorias. El primer algoritmo utilizado para el control de trayectorias debe permitir a los robots navegar suavemente, pasando en orden por una serie de puntos definidos para un comportamiento en particular. Se requiere que el segundo algoritmo sea un planificador que resuelva el problema de evasión de obstáculos para configuraciones impredecibles y en cambio constante. El tiempo de generación debe cumplir con las restricciones impuestas por el sistema de visión y el envío de información a los robots. El planificador debe utilizar la mayor cantidad de información proveniente del sistema de visión y ser independiente de la acción que ejecuten los robots. Ambos algoritmos deben permitir al robot la navegación estable en el tiempo, es decir, sin cambios drásticos entre dos ciclos de control a menos que las circunstancias cambien radicalmente.

# Capítulo 4

## Diseño del Sistema

En este capítulo se presenta el diseño del sistema de IA. Se parte del análisis de la interacción del sistema con el usuario y con los demás sistemas que conforman la arquitectura completa. Posteriormente, para cada una de las funcionalidades planteadas en los requerimientos del sistema de IA se diseña una solución. Las soluciones se describen tomando como referencia los módulos que conforman la arquitectura del sistema y los algoritmos diseñados para este objetivo. Las funcionalidades que resuelve el sistema de IA son: la recepción el sistema de visión y del *referee box*, la visualización del ambiente de pruebas, el control de movimientos, el simulador de movimientos, la detección del movimiento de la pelota, la estrategia de juego y el control de trayectorias.

### 4.1. Arquitectura del sistema de IA

Partiendo de los requerimientos, se define la arquitectura general del sistema de IA y cómo interactúan sus funcionalidades con el sistema de visión, el *referee* box, los robots y el usuario (figura 4.1).

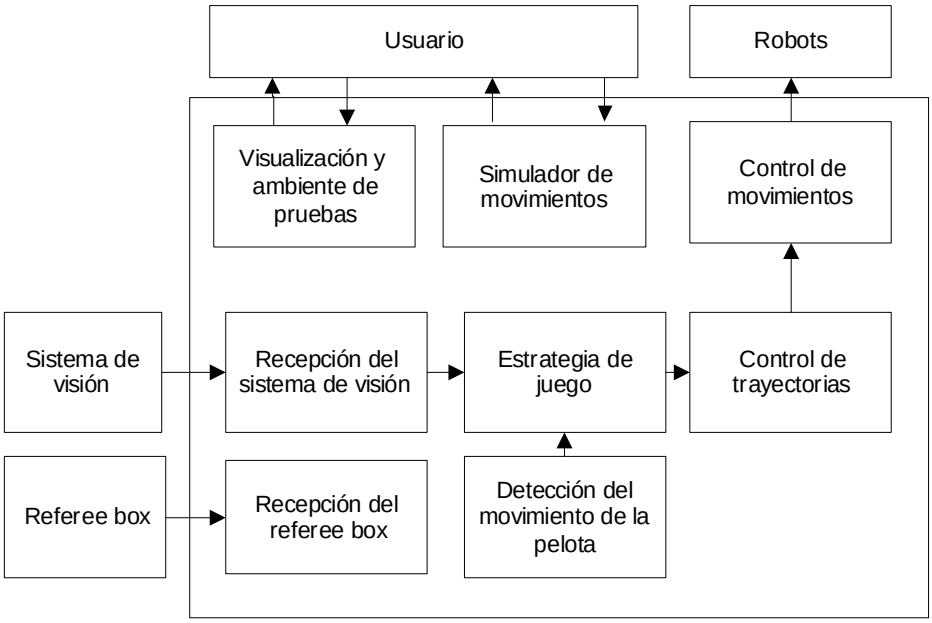



***Figura 4.1. Arquitectura** general del sistema de IA.*

La entradas y salidas del sistema de IA son:

**1.Sistema de visión:** Define la ubicación de los robots y la pelota. Otra forma de obtener esta información es internamente por medio de un simulador.

**2.Referee box:** Envía los comandos (Anexo) utilizados en los partidos a partir de los cuales se generan internamente los estados de juego.

**3.Robots:** Reciben el vector de movimiento y las instrucciones para el uso de sus dispositivos. Existen tres maneras de controlar a los robots: la primera es por medio del módulo de toma de decisiones que incluye la estrategia de juego y la generación de trayectorias; la segunda, por medio de la interfaz de control; la tercera, por algún controlador externo.

**4.Usuario:** Realiza las configuraciones y visualizaciones indispensables para el correcto funcionamiento del sistema. Esta interacción no es necesaria una vez que se desea controlar a los robots de forma autónoma.

La figura 4.2 muestra la arquitectura del sistema de IA, desglosando cada una de sus funcionalidades en todos sus módulos.

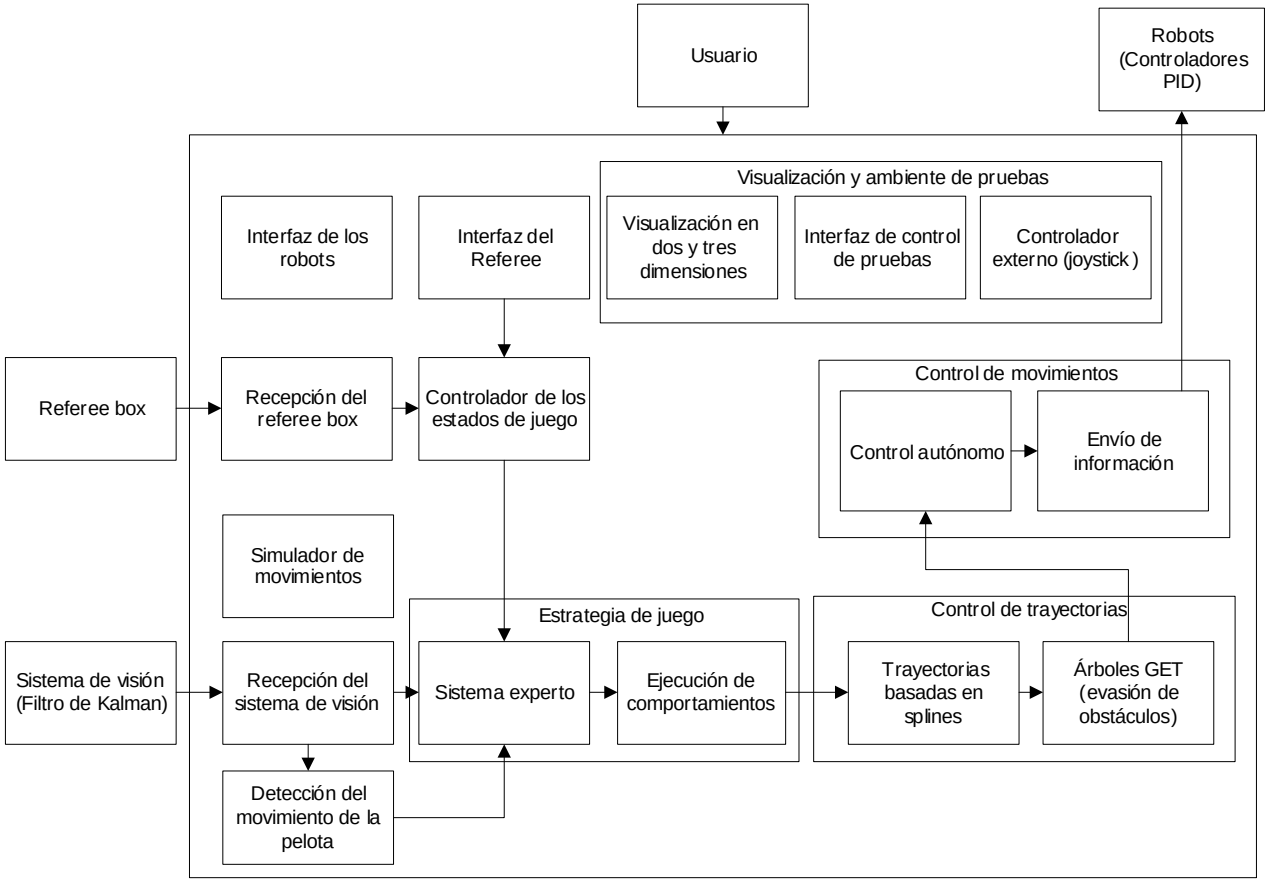



***Figura 4.2.*** *Arquitectura del sistema de IA.*

### 4.2. Recepción del sistema de visión

El sistema de visión realiza el procesamiento de las imágenes que provienen de las dos cámaras de video y entrega al sistema de IA la información referente a la localización de los robots y la pelota tomando como referencia el campo de juego [29]. Esta información se envía en una estructura de datos con la información acerca de la localización de los robots y la pelota. El sistema de visión debe procesar cada imagen a una velocidad mínima de 30 cuadros por segundo. Al estar procesando dos cámaras, envía al menos 60 *sockets* por segundo, considerando que los cuadros de las cámaras se van intercalando.

La información que recibe el sistema de IA a través del sistema de visión es la siguiente:

**Robots del equipo EK (5)**
- Coordenadas *(x, y)* [mm].
- Velocidad de movimiento *(Vx, Vy)* [mm/s].
- Orientación en radianes del frente [rad].
- Velocidad de giro *(Vw)* [rad/s].

**Robots contrarios (5)**
- Coordenadas *(x, y)* [mm].
- Presente en la cancha [booleano].

**Pelota**
- Coordenada *(x, y)* [mm].
- Velocidad de movimiento [mm/s].
- Coordenada de corrección basada en un filtro de Kalman [mm].
- Coordenada de predicción basada en un filtro de Kalman [mm].

### 4.3. Referee box

El *referee box* es el encargado de controlar las acciones en el campo de juego. Este sistema distingue a los equipos por el color del parche central de los robots: amarillo o azul. Por medio de este sistema se reciben los comandos y las notificaciones del partido como goles, tiempos del partido, amonestaciones y tiempos fuera (Anexo).

Las reglas de la liga describen la metodología de un partido y las circunstancias en que se genera cada uno de estos eventos [28]. Un partido tiene una duración de 30 minutos y cada equipo tiene un máximo de cuatro tiempos fuera que no pueden sobrepasar un total de 10 minutos. La tabla 4.1 describe algunas de las reglas más importantes para fines del sistema de IA.

El módulo de recepción de comandos del árbitro es el encargado de recibir los comandos generados por la computadora del árbitro. Estos comandos también pueden ser generados directamente dentro del sistema de IA, por medio de la interfaz del *referee*.

El módulo del controlador de estados de juego traduce estos comandos en estados de juego que son utilizados en la estrategia de juego (figura 4.3). Por ejemplo, COMM_KICKOFF_BLUE se traduce en el estado de juego OUR _KICKOFF cuando el equipo EK utiliza el color azul. La tabla 4.2 muestra los estados de juego y su descripción.

| Situación | Consecuencia |
|---|---|
| Un robot golpea lo suficientemente fuerte a otro robot y altera su libertad de movimiento o lo desplaza en caso de que esté detenido. | Tiro indirecto. |
| Un segundo robot defensor entra al área del portero. | Tiro indirecto. |
| Un robot no respeta la señal de tiro directo o indirecto del equipo contrario. | Tiro directo. |
| Un robot reincide muchas veces en una misma falta. | Amonestación. |
| Un robot arrastra la pelota una distancia mayor de 500 mm. | Tiro indirecto. |
| Al enviarse la señal HALT. | Los robots deben detenerse por completo. |
| Al enviarse la señal STOP. | Los robots deben alejarse 500 mm de la pelota. |
| Un robot toca dos veces la pelota después de cobrar un tiro a favor (tiro directo, indirecto, saque o penalty). | Tiro indirecto. |
| La pelota sale del campo de juego. | Tiro directo (saque lateral, de meta o de esquina). |
| Cuando se marca un penalty o un saque se envía posteriormente un comando READY para indicar que todo está listo. | Si no se cumple esta regla se marca un tiro indirecto. |
| Después de cualquier acción tomada después de enviarse la señal STOP, los equipos deben detectar que la pelota se movió para reiniciar automáticamente su juego. | En caso de que no se reinicie correctamente el juego, se envía un reinicio neutro START. |

***Tabla 4.1.*** *Reglas más importantes durante el partido.*

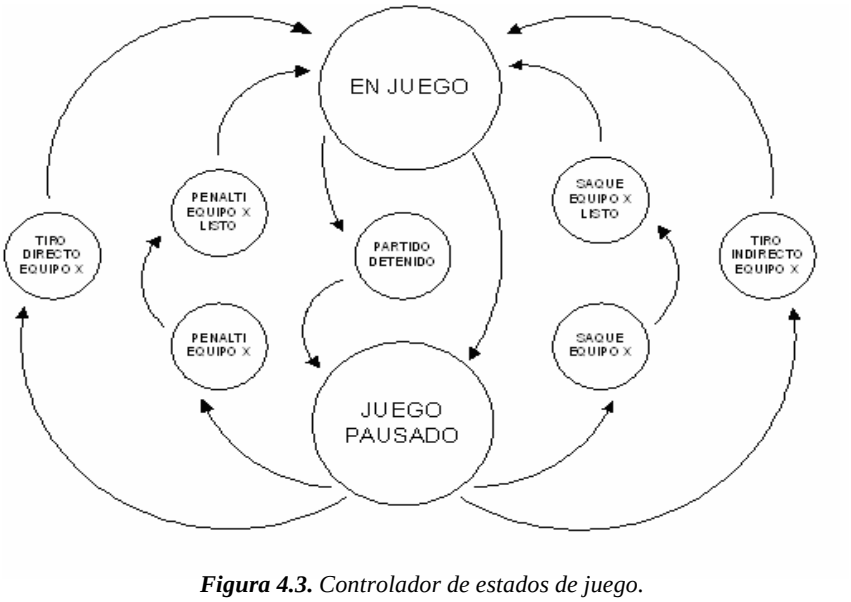



***Figura 4.3.*** *Controlador de estados de juego.*

| Estado de juego | Descripción |
|---|---|
| HALT | Partido detenido |
| STOP | Juego pausado |
| START | Reinicio neutral |
| OUR_PENALTY | Tiro penalty nuestro |
| THEIR_PENALTY | Tiro penalty contrario |
| OUR_PENALTY_READY | Tiro penalty nuestro listo |
| THEIR_PENALTY_READY | Tiro penalty contrario listo |
| OUR_KICKOFF | Saque nuestro |
| THEIR_KICKOFF | Saque contrario |
| OUR_KICKOFF_READY | Saque nuestro listo |
| THEIR_KICKOFF_READY | Saque contrario listo |
| OUR_DIRECT_KICK | Tiro directo nuestro |
| THEIR_DIRECT_KICK | Tiro directo contrario |
| OUR_INDIRECT_KICK | Tiro indicrecto nuestro |
| THEIR_INDIRECT_KICK | Tiro indirecto contrario |

***Tabla 4.2.*** *Estados de juego.*

## 4.4. Visualización y ambiente de pruebas

El sistema de IA cuenta con varios módulos que le permiten realizar las pruebas necesarias tomando en cuenta al sistema de visión y a los robots. Para realizar pruebasen los robots, el sistema cuenta con dos módulos: el módulo del controlador externo y la interfaz de control de pruebas. Para visualizar la información proveniente del sistema de visión, así como la interpretación gráfica de lo que ocurre internamente en el sistema se cuenta con el módulo de visualización en dos y tres dimensiones.

### 4.4.1. Controlador externo (joystick)

El uso de un controlador externo no está permitido durante los partidos; sin embargo, se utiliza en el proceso de preparación de los robots para un juego con el objeto de verificar que los demás sistemas funcionen correctamente. En especial es útil para probar lo siguiente: cambios realizados en la programación del robot y en las piezas del robot, comunicación entre la computadora y los robots libre de interferencia, tiempo de reacción de los robots en diferentes direcciones, funcionamiento correcto de las tarjetas electrónicas y motores, del sistema de control y el pateo de la pelota.

El sistema de IA interactúa con el controlador, interpretando sus movimientos y los eventos en los botones para definir el vector de movimiento de traslación y de giro, así como el uso de sus dispositivos.

### 4.4.2. Interfaz de control de pruebas

La interfaz de control de pruebas, como su nombre lo indica, es útil para realizar pruebas y obtener resultados concretos medibles. Consiste en una interfaz que permite definir manualmente una posición y orientación final en la cancha a una velocidad determinada o respecto del sistema de referencia del robot. También cuenta con una interfaz para controlar por medio del ratón los movimientos del robot. Este método permite mover fácilmente al robot por toda la cancha, validar que el sistema de visión reconozca correctamente al robot en todas partes y validar una transición suave en el límite entre las imágenes de las dos cámaras que ocurre a la mitad de la cancha.

### 4.4.3. Visualización en dos y tres dimensiones

El campo de juego representa el espacio de acción para el sistema de IA. Dentro de este espacio se tiene conocimiento de los robots y de la pelota. El campo de juego está bien definido por las reglas de la liga SSL [28]. Las dimensiones de la cancha y el modelo de referencia del mundo se muestran en la figura 4.4.

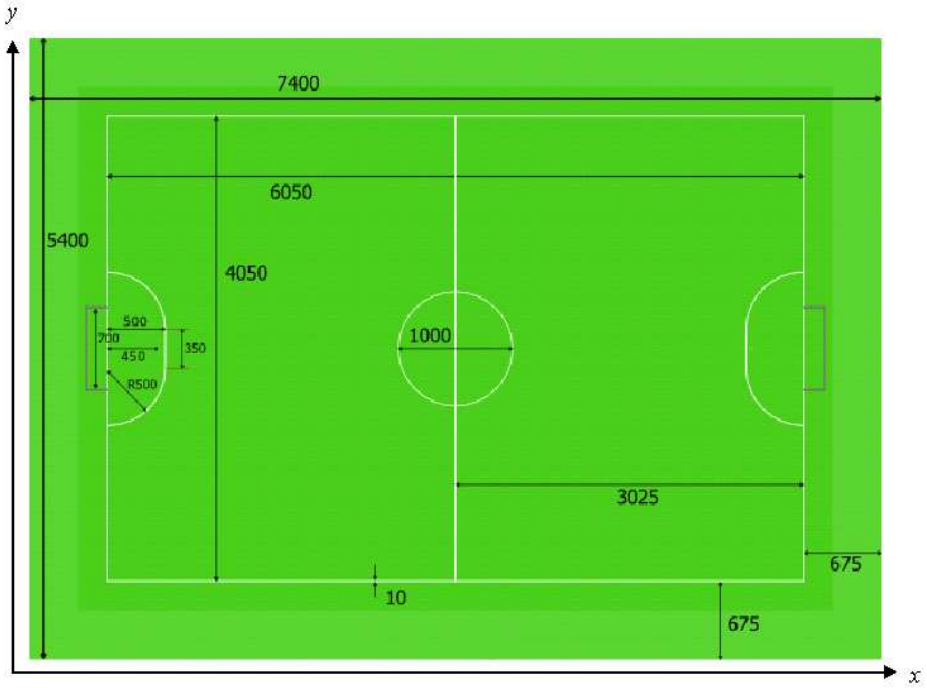



***Figura 4.4.*** *Dimensiones del campo de juego en milímetros.*

El ambiente gráfico del sistema de IA recrea virtualmente lo que ocurre en el campo de juego por medio de la información recibida del sistema de visión. También permite que el usuario conozca el funcionamiento de todos los módulos del sistema; tiene dos formas de desplegar el ambiente de la cancha: en dos y tres dimensiones como lo muestran las figuras 4.5 y 4.6 respectivamente.

En el ambiente gráfico se muestran las posiciones de las líneas, las porterías, los robots y la pelota. Además, permite conocer las rutas que siguen los robots, el despliegue de los árboles utilizados para la planeación de rutas y las trayectorias definidas para los

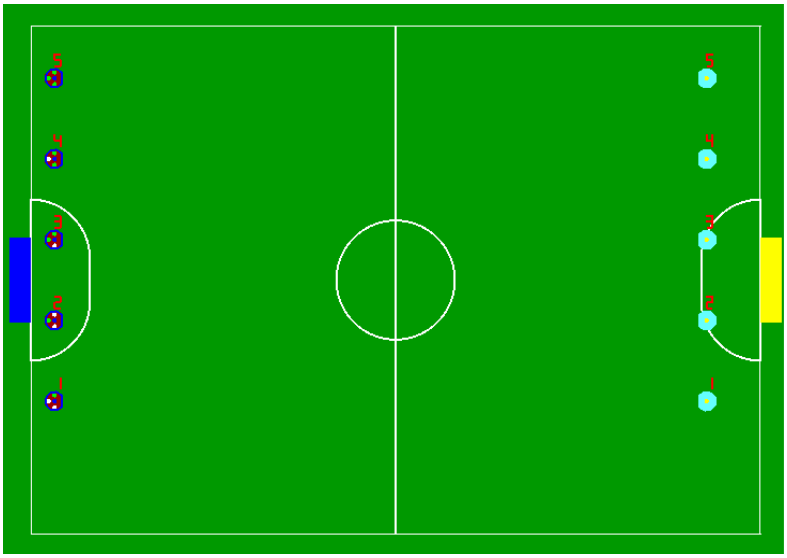


***Figura 4.5.*** *Ambiente gráfico bidimensional.*

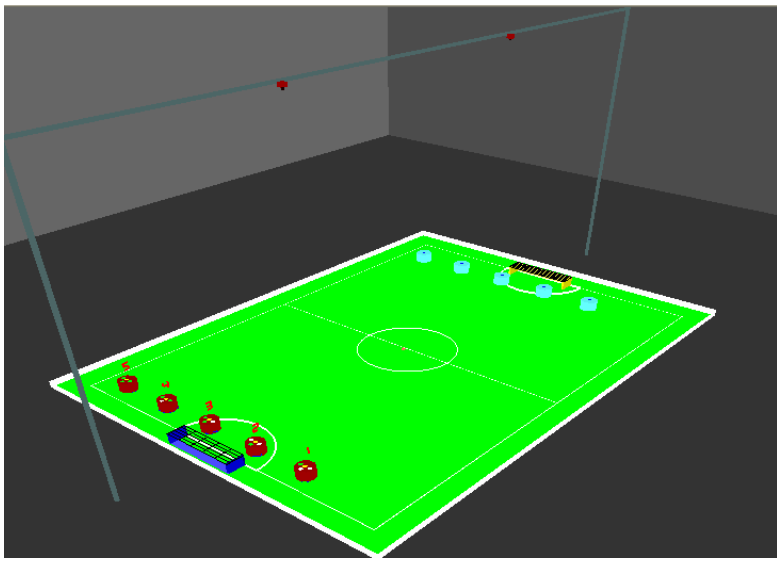


***Figura 4.6.*** *Ambiente gráfico tridimensional.*

robots. También permite mostrar cuándo se activan los dispositivos de los robots, así como la numeración y el patrón de parches de cada robot para ubicarlos fácilmente en la cancha.

El movimiento del robot se compone de una línea que indica la coordenada final y un círculo que representa la orientación final. Un ejemplo de la representación de este movimiento en dos y tres dimensiones se muestra en la figura 4.7, partiendo de un estado actual para llegar a un estado final. Cada estado se compone de una coordenada y un ángulo.

**Estado actual**

- Ángulo actual: 0 grados
- Coordenada actual: (450, 1342) [mm]

**Estado final**

- Ángulo final: 180 grados
- Distancia a moverse desde la coordenada de inicio: 600 mm
- Ángulo de movimiento: 45 grados respecto al sistema de referencia de la cancha
- Coordenada final: (874, 1767) [mm]

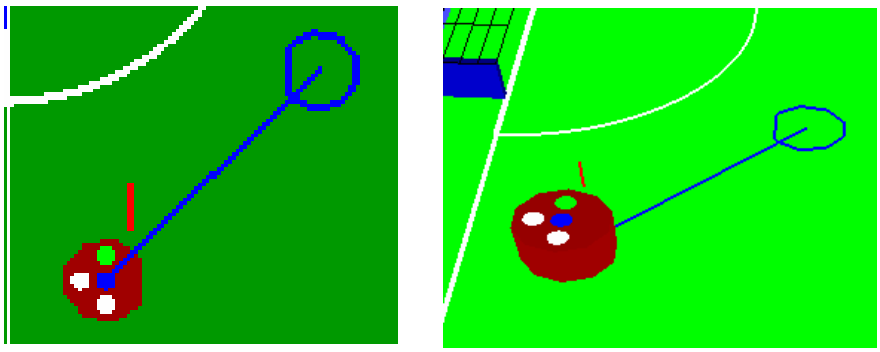


***Figura 4.7.*** *Estado actual y estado final del robot.*

Los niveles de potencia de pateo se indican con un círculo exterior que varía en la gama de los colores amarillo y rojo, en donde amarillo es el nivel 1 y rojo es el nivel 7. La figura 4.8 muestra la representación de los siete niveles de potencia de pateo ordenados en forma ascendente.

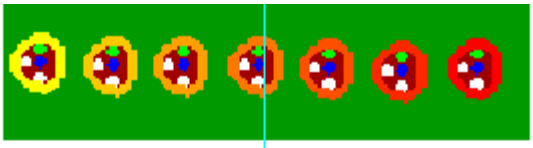


***Figura 4.8.*** *Niveles de potencias de pateo.*

La figura 4.9 muestra la activación del *dribbler* con un círculo morado y la figura 4.10 la representación de un robot con su identificador y patrón de parches.

***Figura 4.9.*** *Activación del dribbler.*

***Figura 4.10.*** *Número del robot y patrón de parches.*

La trayectoria permite ver los últimos *n* puntos por los que ha pasado el robot como lo muestra la figura 4.11. Esta representación es útil para saber si el robot se está moviendo adecuadamente y permite observar la distancia que avanza el robot entre cada medición del sistema de visión.

La pelota se dibuja como un círculo naranja (esfera en tres dimensiones). Además se dibuja una línea que representa la predicción que entrega el sistema de visión por medio del filtro de Kalman (figura 4.12).

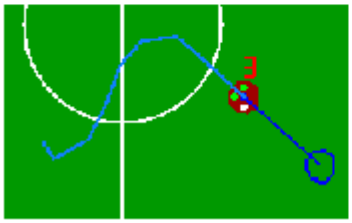


***Figura 4.11.*** *Trayectoria del robot.*

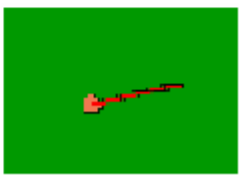


***Figura 4.12.*** *Predicción de la pelota.*

## 4.5. Control de movimientos

El equipo está formado por robots omnidireccionales que se mueven en un espacio bidimensional por la arquitectura del sistema. Durante un ciclo de control se envían constantemente instrucciones a los robots a un mínimo de 60 veces por segundo.

El control de los movimientos del robot se realiza en dos niveles:

**a) Primer nivel:** Consiste en la implementación del movimiento omnidireccional realizado de forma individual en cada uno de los procesadores de los robots. En este modelo se integra el controlador Proporcional Integral Derivativo (PID) y se define la información que recibe cada uno de los robots desde la computadora central.

**b) Segundo nivel:** Consiste en el control realizado de forma centralizada para cada uno de los robots dentro del sistema de IA. En este modelo se incluye un filtro para regular los cambios de velocidades y las acciones necesarias para realizar el control de forma autónoma a partir de la información proveniente del sistema de visión.

### 4.5.1. Movimiento omnidireccional

El movimiento omnidireccional brinda una completa maniobrabilidad de los robots que pueden moverse en cualquier dirección sin requerir una orientación específica. La principal ventaja es que el robot puede girar mientras se mueve en cierta dirección. Para lograr este desplazamiento se utilizan ruedas omnidireccionales (figura 4.13).

***Figura 4.13.*** *Rueda omnidireccional.*

Este diseño permite que cuando una rueda recibe una tracción en la dirección normal al eje del motor pueda deslizarse sin fricción en la dirección de este eje. El robot utiliza cuatro motores colocados de forma simétrica con un valor ö de 38 grados, en donde cada uno ejerce una fuerza dependiendo de su orientación como lo muestra la figura 4.14.

El robot cuenta con tres vectores de movimiento: un vector de traslación compuesto de una magnitud (velocidad a la que deseamos moverlo) y una dirección, representado en la forma cartesiana *Vx*, *Vy*, tomando como referencia el frente del robot y el vector de giro *(w)* que define la velocidad y el sentido en el que rota el robot. La figura 4.15 muestra el sistema del referencia y la 4.16, su alineación respecto al diseño mecánico del robot.

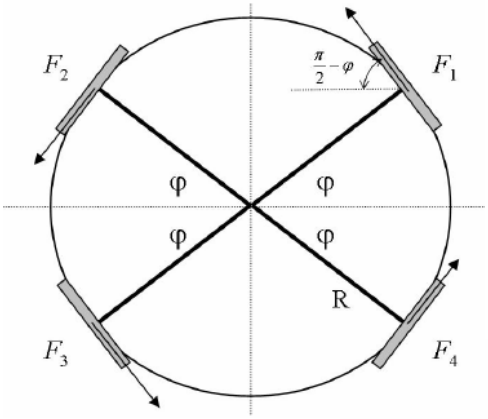



***Figura 4.14.*** *Distribución de fuerzas para los motores colocados en forma simétrica.*

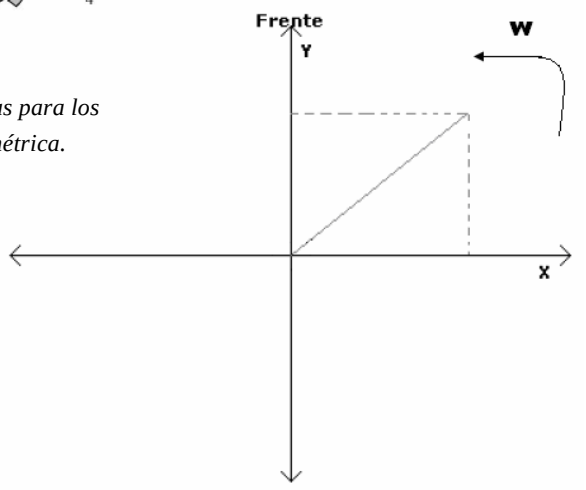



***Figura 4.15.*** *Sistema de referencia para el modelo omnidireccional.*

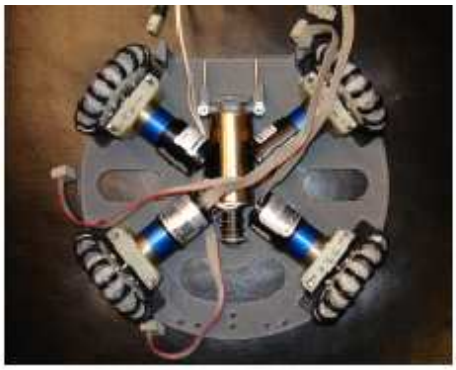


*Figura 4.16. Alineación de los motores respecto al sistema de referencia.*

A partir de este modelo se puede realizar un desarrollo matemático para deducir las correspondencias entre las velocidades de cada vector de movimiento del robot y las velocidades de los motores. El sistema de ecuaciónes 4.1 describe esta relación, en donde $R$ es el radio del robot (90 mm), $w$ es el vector de giro, $Vx$, $Vy$, representan el vector de traslación, $\varphi$ es la orientación de los motores, $v_1$, $v_2$, $v_3$, $v_4$ corresponden a la velocidad de cada uno de los motores. Lo anterior conforme al modelo de la figura 4.14 [30].

$$(v_1, v_2, v_3, v_4)^{\mathrm{T}} = \begin{pmatrix} \sin\varphi & \cos\varphi & 1 \\ -\sin\varphi & -\cos\varphi & 1 \\ \sin\varphi & -\cos\varphi & 1 \\ \sin\varphi & \cos\varphi & 1 \end{pmatrix} \begin{pmatrix} v_x \\ v_y \\ R\omega \end{pmatrix} \tag{4.1}$$

El sistema de ecuaciones 4.1 puede representarse de la siguiente forma:

*(4.2)*

En donde $m$ es el vector que contiene las velocidades de los motores, $v$ es el vector que contiene las velocidades de cada vector de movimiento del robot y $D$ es la matriz de transición.

Partiendo de la ecuación 4.2 es posible formular el proceso inverso para obtener las velocidades de cada vector de movimiento a partir de las velocidades de los motores:

$$m = Dv \tag{4.3}$$

La matriz $D^+$ es la pseudoinversa de la matriz $D$ y puede expresarse en función del ángulo en el que están colocados los motores:

$$D^+ = \frac{1}{4}\begin{pmatrix} -\frac{1}{\sin\varphi} & -\frac{1}{\sin\varphi} & \frac{1}{\sin\varphi} & \frac{1}{\sin\varphi} \\ \frac{1}{\cos\varphi} & -\frac{1}{\cos\varphi} & -\frac{1}{\cos\varphi} & \frac{1}{\cos\varphi} \\ 1 & 1 & 1 & 1 \end{pmatrix} \tag{4.4}$$

La ecuación 4.1 muestra que cuando los vectores de traslación son cero, pero el vector de giro es diferente de cero, el resultado será que todos los motores se muevan en el mismo sentido y magnitud, dependiendo directamente de la magnitud y el sentido del vector de giro. En cambio, si el vector de giro es cero y los vectores de traslación toman un valor distinto de cero, las velocidades y sentidos de los motores dependen de su orientación, así como de la magnitud y sentido de los vectores *Vx*, *Vy*. Finalmente, el modelo omnidireccional permite que un robot se traslade y gire al mismo tiempo cuando los tres vectores son diferentes de cero.

### 4.5.2. Envío de información

En el ámbito de las telecomunicaciones, un protocolo es un conjunto de reglas que deben ser respetadas para que pueda ser realizado un proceso de comunicaciones [6]. Es necesario establecer estas reglas para que la comunicación entre el sistema de IA y los robots se lleve a cabo de forma confiable.

El módulo de envío de información escala los vectores de velocidad de traslación y giro para definir 256 niveles de velocidad para lo cual se requieren 8 bits. Además, define el uso del sistema de pateo y del sistema de control de la pelota. Cada robot recibe los tres vectores de movimiento necesarios para el modelo omnidireccional que para su envío son descompuestos en su magnitud y signo. Se utiliza también un bit para el control de la pelota que indica la activación del motor que controla la barra del *dribbler*, que sólo puede girar en un sentido para atrapar la pelota por lo que no requiere el envío de su sentido ni de su magnitud. Para el pateo de la pelota se tomaron 3 bits con la finalidad de contar con los siete diferentes niveles de pateo y el cero que indica que el robot puede reiniciar la carga de sus capacitores para el siguiente pateo. En conjunto, en cada ciclo de control, cada uno de los robots recibe la información descrita en la tabla 4.3.

| Bloque | Descripción | Cantidad de información |
|---|---|---|
| *Control* | Control de la pelota | 1 bit |
| | Pateo de la pelota | 3 bits |
| | Signo vector *x* | 1 bit |
| | Signo vector *y* | 1 bit |
| | Signo vector *w* | 1 bit |
| | Reservado | 1 bit |
| *Velocidades* | Magnitud vector *x* | 8 bits |
| | Magnitud vector *y* | 8 bits |
| | Magnitud vector *w* | 8 bits |

*Tabla 4.3. Estructura del paquete de información para cada robot.*

Se puede asumir que si se reserva un byte para el bloque de control, entonces el tamaño del paquete por robot se puede calcular completamente en bytes utilizando la ecuación 4.5.

$$P = m + 1 \text{ bytes} \tag{4.5}$$

**En donde:**

$P$: Tamaño del paquete por robot.

$m$: Número de vectores de movimiento que tiene el robot.

Cada paquete debe ser enviado a cada uno de los robots con su información correspondiente, de esta manera se forma una trama que debe contener un bloque de control general para especificar su tamaño y otro bloque en donde se integren los cinco paquetes de los robots como lo muestra la tabla 4.4.

| Control general | Robot 1 | Robot 2 | Robot 3 | Robot 4 | Robot 5 |
|---|---|---|---|---|---|
| 1 byte | *P* bytes | *P* bytes | *P* bytes | *P* bytes | *P* bytes |

*Tabla 4.4. Estructura de la trama del módulo de comunicación.*

El tamaño de la trama es de:

$$T = 5P + 1 \text{ bytes} \qquad (4.6)$$

**En donde:**
*T:* Tamaño de la trama.

La ecuación 4.7 se obtiene de sustituir la ecuación 4.5 en 4.6:

$$T = 5m + 6 \text{ bytes} \qquad (4.7)$$

En total se envían 21 bytes en cada ciclo de control considerando los tres vectores de movimiento y que el sistema trabaja entre 60 y 90 ciclos por segundo.

### 4.5.3. Controladores PID

Para resolver el problema del control a nivel de los robots se recurre a un controlador PID para cada uno de los vectores de movimiento. Anteriormente se utilizaba este controlador para cada uno de los motores [14]. La principal ventaja de este cambio es que los controladores trabajan a nivel global, relacionando directamente la respuesta del robot al movimiento indicado por el sistema de IA, en lugar de que cada motor haga la corrección de forma independiente sin tomar en consideración el movimiento de los demás motores, mejorando en consecuencia las trayectorias en línea recta.

Un controlador PID requiere un valor de referencia *(Rf)* y un valor de retroalimentación *(Fd)* para entregar un valor de salida *(Out)*. Al utilizar tres controladores PID, uno para cada vector de movimiento, se requiere un valor de referencia para cada controlador *(VxRf, VyRf, wRf)* siguiendo el modelo de referencia de la figura 4.15. Estos valores se obtienen directamente al leer la información que envía el sistema de IA por medio del transmisor inalámbrico y representan la velocidad a la que se desea mover el robot. Los valores de retroalimentación *(VxFd, VyFd, wFd)* no se pueden obtener directamente, ya que lo único que se conoce es la velocidad de retroalimentación de cada motor a partir de sus *encoders (m1Fd, m2Fd, m3Fd, m4Fd)* [14]. Por medio de la ecuación 4.3 es posible obtener los vectores de movimiento (valores de retroalimentación) a partir de la velocidad de cada uno de los motores.

Las figuras 4.17 y 4.18 muestran el proceso utilizado para integrar la información requerida por cada uno de los controladores PID y de esta forma definir los tres vectores de salida *(VxOut, VyOut, wOut)*. Una vez obtenidas las velocidades de salida para cada vector *(VxOut, VyOut, wOut)*, se descomponen, utilizando la ecuación 4.2, para obtener las velocidades de cada motor *(m1Out, m2Out, m3Out, m4Out)* que se aplican a cada uno de los motores en forma de señales moduladas en ancho de pulso (PWM) [14]. La asignación de cada motor y su sentido de giro siguen el sistema de referencia presentado en la figura 4.14 en donde las fuerzas *F1, F2, F3 y F4* corresponden a las ejercidas por el motor correspondiente *m1, m2, m3, m4.*

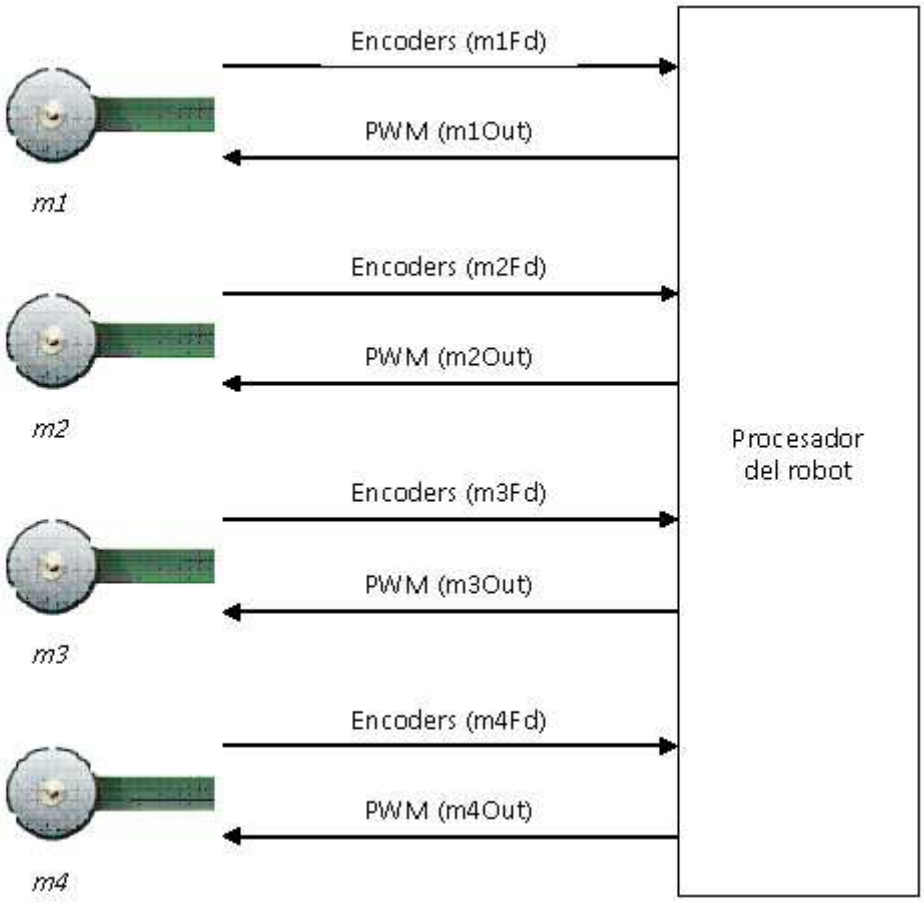



***Figura 4.17.*** *Modelo de control de cada motor por medio del procesador del robot.*

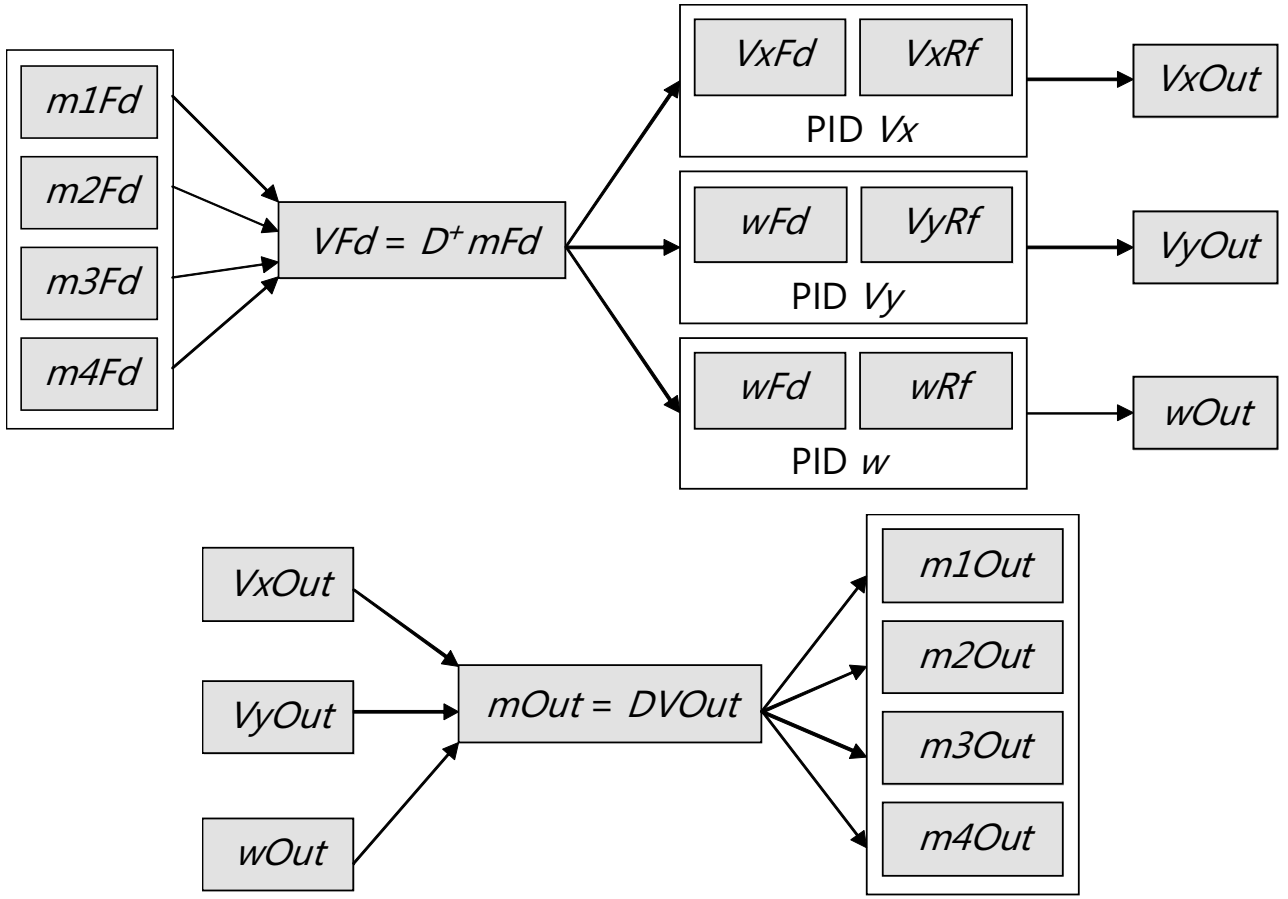



***Figura 4.18.*** *Modelo global de corrección PID para el control omnidireccional.*

*m1Fd:* Velocidad de retroalimentación del motor 1.
*m2Fd:* Velocidad de retroalimentación del motor 2.
*m3Fd:* Velocidad de retroalimentación del motor 3.
*m4Fd:* Velocidad de retroalimentación del motor 4.
*m1Out:* Velocidad aplicada al motor 1.
*m2Out:* Velocidad aplicada al motor 2.
*m3Out:* Velocidad aplicada al motor 3.
*m4Out:* Velocidad aplicada al motor 4.
*VxFd:* Velocidad de retroalimentación en el eje *x* del robot.
*VyFd:* Velocidad de retroalimentación en el eje *y* del robot.
*wFd:* Velocidad de retroalimentación en el eje *w* del robot (giro).
*VxRf:* Velocidad de referencia en el eje *x* del robot.
*VyRf:* Velocidad de referencia en el eje *y* del robot.
*wRf*: Velocidad de referencia en el eje 2 del robot (giro).
*VxOut:* Velocidad aplicada en el eje *x* del robot.
*VyOut:* Velocidad aplicada en el eje *y* del robot.
*wOut:* Velocidad aplicada en el eje *w* del robot.
*D+:* Matriz utilizada para obtener la velocidad del robot en función de la velocidad de cada motor.
*D:* Matriz utilizada para obtener la velocidad de cada motor en función de la velocidad del robot.
PID *Vx:* Filtro PID aplicado a la velocidad en el eje *x* del robot.
PID *Vy:* Filtro PID aplicado a la velocidad en el eje *y* del robot.
PID *w:* Filtro PID aplicado a la velocidad en el eje *w* del robot (giro).

La forma en que responden los robots depende del valor de las constantes *Kp, Ki, Kd,* para cada uno de los tres controladores PID, las cuales pueden ser ajustadas manualmente o por medio de un aprendizaje por refuerzo [16].

El modelo de corrección global permite detectar y corregir el derrape del robot cuando la rotación de alguna de sus ruedas no es consistente [30].

### 4.5.4. Control autónomo

Un problema que se puede presentar es la variación brusca de los vectores de movimientos enviados al robot. Partiendo de un ejemplo en donde el robot está detenido y deseamos

que se mueva rápidamente en cierta dirección, los controladotes PID realizarán una corrección a este modelo, sin embargo, es conveniente filtrar estos cambios para que el robot los realice de una forma suave. Para cada uno de los vectores de movimiento del robot se define una tasa de cambio. Para ejemplificar esto se parte del planteamiento descrito en la tabla 4.5, en donde se desea mover al robot hacia atrás durante un segundo y, posteriormente, hacia el frente durante dos segundos siguiendo el sistema de referencia de la figura 4.15.

**Ejemplo:**

| Vector | *Vx* | *Vy* | *w* |
|---|---|---|---|
| $T_0 = 0$ | 0 | –2000 mm/s0 | 0 |
| $T_1 = 1s$ | 0 | 2000 mm/s | 0 |
| $T_2 = 3s$ | 0 | 0 mm/s | 0 |

***Tabla 4.5.*** *Velocidades deseadas en el tiempo.*

Suponiendo una tasa de cambio de 2500 mm/s$^2$ (1250/500 ms$^2$), la descomposición de estos movimientos para filtrar los cambios en el tiempo se realiza conforme a la tabla 4.6.

Este cambio se aplica de la misma forma para el movimiento lateral *(Vx)* y para el vector de giro *(w)* con la diferencia de que la tasa de cambio de este último se define en radianes por segundo. Como resultado se obtiene el filtrado de todos los cambios en cada vector de movimiento enviado a los robots.

El caso del frenado se debe considerar de manera independiente, ya que no es posible esperar hasta que el robot llegue a su meta para enviarle una velocidad de cero en alguno de sus vectores de movimiento, porque si el robot se mueve muy rápido oscilaría alrededor de su coordenada final u orientación final. Para solucionar esto, manteniendo la misma tasa de cambio, se disminuye la velocidad dependiendo de la distancia que hay entre el robot y la meta. De esta forma, cuando el robot está muy cerca de la meta nos aseguramos de que su velocidad sea lo suficientemente baja para poder detenerlo. Lo mismo se aplica para el vector de giro, pero disminuyendo su velocidad en función de la diferencia entre su orientación y la orientación final.

Una vez definido el modelo de movimiento, utilizando la información recibida por el sistema de visión, se puede realizar el control de los robots de forma autónoma. Para esto se define la velocidad que se va a enviar al robot en cada uno de sus vectores de movimiento realizando las siguientes acciones:

**Ejemplo:**

| Tiempo (segundos) | *Vy* deseado (mm/s) | *Vy* filtrado (mm/s) |
|---|---|---|
| $T_0 = 0$ | 0 | 0 |
| 0.25 | −2000 | −625 |
| 0.5 | −2000 | −1250 |
| 0.75 | −2000 | −1875 |
| $T_1 = 1$ | –2000 | −2000 |
| 1.25 | 2000 | −1375 |
| 1.5 | 2000 | − 750 |
| 1.75 | 2000 | − 125 |
| $T_2 = 2$ | 2000 | 500 |
| 2.25 | 2000 | 1125 |
| 2.5 | 2000 | 1750 |
| 2.75 | 2000 | 2000 |
| 3 | 2000 | 2000 |

***Tabla 4.6.*** *Modelo de aceleración para un vector de movimiento.*

1. Filtrar por el modelo de aceleración y desaceleración la velocidad definida por el módulo de control de pruebas, el controlador externo o la toma de decisiones (figura 4.2).
2. Frenar cuando el robot se encuentre próximo a su coordenada final, disminuyendo la velocidad de traslación.
3. Frenar cuando el robot se encuentre próximo a su orientación final, disminuyendo su velocidad de giro.
4. Definir cuál es el sentido más corto en el que debe girar el robot.
5. Detener al robot cuando haya alcanzado su coordenada final u orientación final (puede dejar de trasladarse y seguir girando o dejar de girar y trasladarse hasta llegar a su coordenada final).

### 4.6. Simulador de movimientos

El simulador tiene como finalidad actualizar la posición y la velocidad de los robots y la pelota, tomando como referencia un modelo dinámico. Se utiliza cuando no se tiene información del sistema de visión.

Para realizar el modelo de simulación se tomó como referencia la teoría del movimiento omnidireccional que parte de las velocidades de cada uno de los tres vectores de movimiento *Vx*, *Vy*, *w* [32]. De esta forma, la posición del robot se actualiza dentro de un *timer*, partiendo de los vectores enviados al robot en su sistema de referencia (figura 4.15), convertidos al sistema de referencia del mundo (figura 4.4). Las siguientes ecuaciones muestran cómo se obtienen la coordenada y orientación del robot ($X_t$, $Y_t$, $W_t$) en función del estado anterior ($X_{t-1}$, $Y_{t-1}$, $W_{t-1}$), de su velocidad de movimiento y del periodo del *timer Ts*. Un rango de valores típicos para el periodo de simulación son entre 1/30 y 1/60 segundos. Este rango de valores está relacionado con el periodo de renderización del ambiente gráfico y la frecuencia de muestreo del ojo humano.

$$\begin{aligned} X_t &= X_{t-1} + Vx\ Ts \\ Y_t &= Y_{t-1} + Vy\ Ts \\ W_t &= W_{t-1} + w\ Ts \end{aligned} \qquad (4.8)$$

En la simulación se toman en consideración los modelos de aceleración y desaceleración aplicados a los robots y factores como la desaceleración de la pelota por la fricción con el campo de juego y por los rebotes con los bordes de la cancha.

El uso del simulador permite observar la reacción de los robots ante diferentes comportamientos y cómo las diferentes condiciones de juego afectan la toma de decisiones. También es extremadamente útil para probar diferentes algoritmos de navegación, ya que si la respuesta del robot en un ambiente simulado no es la esperada, resulta mucho más complicado que los robots en el ambiente real se muevan correctamente.

Es difícil realizar un modelo de simulación en el que se tomen en cuenta todos los factores originados en el mundo real como son: la respuesta de los motores, la fricción con el campo de juego, el modelo de control utilizado en el robot, el error de medición del sistema de visión y los retrasos de procesamiento del sistema completo.

La principal razón por la que un robot simulado no puede ser controlado en forma autónoma es que las variaciones existentes entre lo que sucede en el mundo real y el mundo simulado se acumulan en el tiempo ocasionando un desfase. El resultado es que el sistema de IA cree que el robot tiene cierta posición y orientación en el campo de juego cuando realmente tiene otras. Para sincronizar el modelo con el mundo real se utiliza el sistema de visión global. La figura 4.19 muestra tres etapas de la simulación del movimiento del robot.

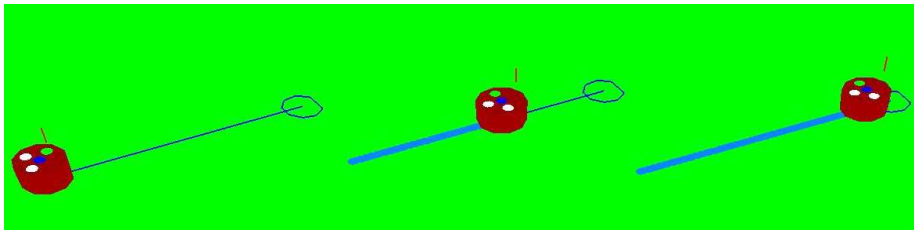


***Figura 4.19.*** *Simulación del movimiento del robot.*

## 4.7. Detección del movimiento de la pelota

En ocasiones para diseñar los comportamientos, no es suficiente conocer únicamente la posición de la pelota, sino también su dirección de movimiento; por ejemplo, cuando se desea interceptar la pelota, recibir un pase o bloquear un tiro del equipo contrario.

El principal problema es que el movimiento de la pelota no se conoce a partir del procesamiento de una sola imagen. Para resolverlo, se puede tomar la información del ciclo anterior y asumir que el movimiento de la pelota se mantendrá igual para el siguiente ciclo deduciendo su vector de movimiento; sin embargo, una mejor forma de solucionarlo es mediante un filtro recursivo discreto como el de Kalman, el cual permite que las predicciones tomen en consideración el error con respecto a la medición y así formular una corrección utilizada en la siguiente predicción, minimizar el error de covarianza.

Para realizar la predicción es necesario realizar un modelo del movimiento de la pelota a partir de su velocidad de movimiento y posición en la cancha. Partiendo de las cuatro

variables *vx, vy, x, y,* se diseñan las ecuaciones dinámicas para poder obtener el estado $X_{k,}$ partiendo del estado anterior $X_{k-1}$, asumiendo que la pelota se mueve en un espacio libre y su velocidad tiende a disminuir por una constante de fricción *l.* Este proceso se realiza dentro de un lapso de tiempo definido por $\Delta t$ de la siguiente forma:

$$\begin{aligned} x_k &= x_{k-1} + vx_k \Delta t \\ y_k &= y_{k-1} + vy_k \Delta t \\ vx_k &= vx_{k-1}\, ly \\ vy_k &= vy_{k-1}\, l \end{aligned} \qquad (4.9)$$

Es posible obtener la constante de fricción en función del número de estados a futuro y de esta forma definir una generalización del modelo planteado en el sistema de ecuaciones 4.9 para obtener la posición de la pelota en *n* estados posteriores partiendo del estado inicial *k-1.*

$$\begin{aligned} x_{k-1+n} &= x_k + vx_k \Delta t\, l^n \\ y_{k-1+n} &= y_k + vy_k \Delta t\, l^n \end{aligned} \qquad (4.10)$$

La constante de fricción extendida a *n* estados adelante se puede obtener por medio de la repetición del proceso definido en el sistema de ecuaciones 4.9, tomando como ejemplo la velocidad de movimiento en el eje *x* de la cancha (figura 2.8).

**Ejemplo:**

| n | $Vx_{k-1+n}$ (en función del estado anterior) | $Vx_{k-1+n}$ (en función del estado inicial) |
|---|---|---|
| 1 | $VX_{k-1}\, l$ | $VX_{k-1}\, l$ |
| 2 | $VX_k\, l$ | $VX_{k-1}\, l^2$ |
| 3 | $VX_{k+1}\, l$ | $VX_{k-1}\, l^3$ |

El siguiente paso es representar el sistema de ecuaciones 4.9 en términos de la matriz de transición A de acuerdo a la ecuación de predicción del filtro de Kalman:

$$\begin{aligned} X_k &= [x_k\, y_k\, vx_k\, vy_k] \\ X_{k-1} &= [x_{k-1}\, y_{k-1}\, vx_{k-1}\, vy_{k-1}] \end{aligned} \qquad (4.11)$$

$$
\begin{aligned}
X_k &= AX_{k-1} + BU_k \\
A &= [1\ 0\ \Delta t\ 0] \\
&\quad [0\ 1\ 0\ \Delta t] \\
&\quad [0\ 0\ l\ 0] \\
&\quad [0\ 0\ 0\ l]
\end{aligned}
\qquad (4.12)
$$

De esta forma, en el modelo de predicción definido en la ecuación 4.12, suponemos que el parámetro opcional de control $BU_k$ es cero.

Las mediciones del sistema consisten en la lectura de las posiciones y velocidades de la pelota. Estos valores se asignan al vector $Z_k$ y se obtienen directamente por la localización en el sistema de visión.

$$
\begin{aligned}
Z_k = &\ [xm_k] \\
&\ [ym_k] \\
&\ [vxm_k] \\
&\ [vym_k]
\end{aligned}
\qquad (4.13)
$$

Finalmente, es necesario definir el ruido asociado a cada una de las variables en los ruidos de predicción *(Q)* y de medición *(R)*. Estos parámetros representan cuánta confianza se tiene en el modelo utilizado para la predicción y cuánta en las mediciones obtenidas.

A continuación se describen los pasos del algoritmo:

**1. Medición**: Se calculan la velocidad y posición de la pelota y se asignan al vector $Z_k$.

**2. Predicción**: Se utiliza el modelo de predicción para obtener el estado actual a partir del anterior. Como este paso es recursivo, se toma la información de la última corrección. En la primera iteración no existe información y el vector $X_{k-1}$ se inicializa en ceros.

$$
\begin{aligned}
XP_k &= A\,X_{k-1} \\
PP_k &= A\,P_{k-1}\,A' + Q
\end{aligned}
\qquad (4.14)
$$

**3. Corrección**: Se realiza la corrección para el estado *k*.

$$K_k = PP_k\, H\,' 1 / ((H\, PP_k)\, H\,' + R)$$

$$XU_k = XP_k + K_k\,(Z_k\, H\, XP_k) \qquad (4.15)$$

$$PU_k = (\mathrm{I} - K_k\, H)\, PP_k$$

Se asigna el estado $k - 1$ para que pueda ser utilizado en la siguiente iteración:

$$X_{k-1} = XU_k$$

$$P_{k-1} = PU_k \qquad (4.16)$$

Las variables que se ajustaron al modelo de movimiento de la pelota son:

$X_{k-1}$ : Vector que define la velocidad y posición de la pelota en el estado *k-1*.

$XP_k$ : Vector que defi ne la predicción de la velocidad y la posición de la pelota en el estado *k* a partir del estado *k-1*.

$XU_k$ : Vector que define la corrección de la velocidad y la posición de la pelota en el estado *k*.

*A:* Matriz de transición para realizar la predicción del estado de la pelota.

*A':* Transpuesta de la matriz *A*.

*Q*: Matriz que define el ruido en el proceso de la predicción.

*R:* Matriz que define el ruido en el proceso de la medición.

$Z_k$: Vector que define la medición en el estado *k*.

Las variables utilizadas de forma general por el filtro de Kalman son:

$K_k$: Matriz que define la ganancia de Kalman.

*PPk:* Covarianza del error de la estimación *a priori*.

*PUk:* Covarianza del error de la estimación *aposteriori*.

*I:* Matriz identidad.

*H:* Matriz que relaciona el estado actual con la medición.

## 4.8. Estrategia de juego

El módulo de estrategia de juego se desarrolla con la finalidad de definir y realizar comportamientos autónomos. Además de ser indispensable durante los partidos, permite probar otros comportamientos simples que son útiles para verificar que el sistema completo funcione adecuadamente. De esta forma se prueba el control del robot junto con el sistema de visión, la precisión con la que se aproxima a algún punto determinado, los tiempos de aceleración y frenado, así como la suavidad de los movimientos. El módulo de la estrategia de juego interactúa con dos módulos subsecuentes: la ejecución de comportamientos y el control de trayectorias (figura 4.20).

Antes de realizar una estrategia de juego se definen y programan comportamientos independientes. Para la coordinación de los robots se cuenta con cinco roles distintos: portero, defensa, delantero1 (D1), delantero2 (D2) y delantero3 (D3).

La estrategia de juego consta de dos etapas. La primera se basa en definir las acciones que realizan en equipo los cinco robots y se basa en un sistema experto. La segunda consiste en definir la ejecución del comportamiento de cada uno de los robots en función de la acción definida en la primera etapa.

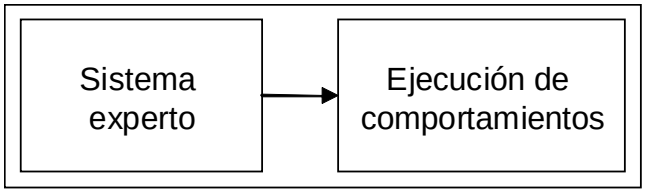



***Figura 4.20.*** *Módulos involucrados en la estrategia de juego.*

### 4.8.1. Sistema experto

El sistema experto se utiliza para definir las acciones que en equipo realizan los robots. La ventaja de utilizar un modelo basado en un sistema experto como CLIPS [11] es que permite programar intuitivamente las reglas que van a definir las acciones que puede realizar un robot, basándose en hechos deducidos a partir de la información de lo que sucede en el campo de juego. El proceso se realiza en tres etapas la definición de las condiciones, los comportamientos y las reglas para cada uno de los robots.

#### 4.8.1.1. Interpretación de los datos

Para diseñar las reglas del sistema experto se parte de un conjunto de condiciones obtenidas a partir del controlador de los estados de juego y la información proveniente del sistema de visión. Los hechos que se utilizan a partir del controlador de los estados de juego se mencionan en la tabla 4.7.

```
(assert (estado_juego Halt))
(assert (estado_juego EdoStop))
(assert (estado_juego EdoStart))
(assert (estado_juego EdoTheirKickoff))
(assert (estado_juego EdoOurKickoff))
(assert (estado_juego EdoTheirPenalty))
(assert (estado_juego EdoOurPenalty))
(assert (estado_juego EdoTheirDirectKick))
(assert (estado_juego EdoOurDirectKick))
(assert (estado_juego EdoTheirIndirectKick))
(assert (estado_juego EdoTheirPenaltyReady))
(assert (estado_juego EdoOurPenaltyReady))
(assert (estado_juego EdoTheirKickoffReady))
(assert (estado_juego EdoOurKickoffReady))
```

***Tabla 4.7.*** *Posibles inserciones dependiendo del estado de juego.*

A partir de la información proveniente del sistema de visión se generan las evaluaciones que ayudan a definir qué comportamiento debe seguir cada robot. Estas evaluaciones toman información de bajo nivel como las coordenadas de los robots y la pelota para obtener información de alto nivel que ayude a definir la estrategia de juego como se muestra en la tabla 4.8.

| Condición | Descripción |
|---|---|
| (region_pelota DentroAreaPorteriaEK) | Pelota dentro del área propia |
| (region_pelota DentroAreaPorteriaVS) | Pelota dentro del área contraria |
| (region_pelota Abajo_port_vs) | Pelota arriba de la portería contraria |
| (region_pelota Arriba_port_vs) | Pelota debajo de la portería contraria |
| (d3_comp BarreraPorteriaDer) | Delantero 3 cubre la portería desde el punto derecho. |
| (d3_comp BarreraPorteriaIzq) | Delantero tres cubre la portería desde el punto izquierdo |
| (def_comp BarreraPorteriaIzq) | Defensa cubre la portería desde el punto izquierdo |
| (zona_pel Defensiva)<br>(rol_pelota D1) | La pelota se encuentra antes de la mitad<br>Delantero 1 es el más cercano a la pelota |
| (rol_pelota D2) | Delantero 2 es el más cercano a la pelota |
| (rol_pelota D3) | Delantero 3 es el más cercano a la pelota |
| (recibe_adel_d1 Si) | Delantero 1 puede recibir un despeje |
| (recibe_adel_d2 Si) | Delantero 2 puede recibir un despeje |
| (recibe_adel_d3 Si) | Delantero 3 puede recibir un despeje |
| (marcar_prim Si) | El robot puede marcar a un contrario |
| (pel_adelante Si) | La pelota está delante de la media cancha |
| (recibe_pase_d1 Si) | Delantero 1 puede recibir un pase central |
| (recibe_pase_d2 Si) | Delantero 2 puede recibir un pase central |
| (recibe_lat_d1 Si) | Delantero 1 puede recibir un pase lateral |
| (recibe_lat_d2 Si) | Delantero 2 puede recibir un pase lateral |
| (libre1 D1) | Delantero 1 es el robot más libre para recibir un pase |
| (libre1 D2) | Delantero 2 es el robot más libre para recibir un pase |
| (libre1 D3) | Delantero 3 es el robot más libre para recibir un pase |
| (d1_adel_mitad Si) | Delantero 1 está delante de la media cancha |
| (d2_adel_mitad Si)<br>(d3_adel_mitad Si) | Delantero 2 está delante de la media cancha<br>Delantero 3 está delante de la media cancha |

***Tabla 4.8.*** *Condiciones para el estado de juego.*

Figura 4.19. Regiones de la pelota en el lado ofensivo.



La figura 4.21 ilustra las regiones utilizadas para definir la posición de la pelota respecto de la portería contraria: `(region_pelota DentroAreaPorteriaVS)`, `(region_pelota Abajo_port_vs)`, `(region_pelota Arriba_port_vs)`.

En cada ciclo de control se evalúa la región en donde se encuentra la pelota y se inserta el hecho correspondiente al resultado de la evaluación.

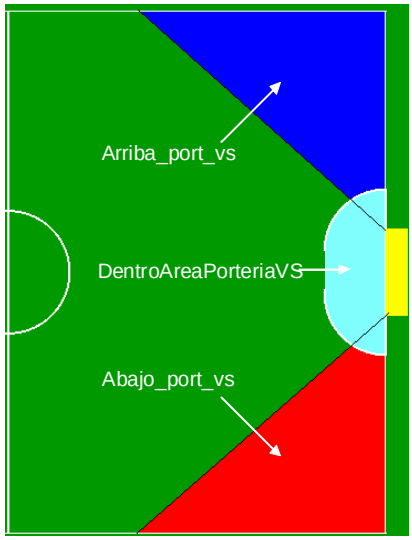



***Figura 4.21.*** *Regiones de la pelota en el lado ofensivo.*

#### 4.8.1.2. Planificación

Los posibles comportamientos que pueden realizar los cinco robots durante un partido dependiendo de la planeación de la estrategia de juego se ejemplifican en la tabla 4.9.

Para ejemplificar la forma en que se realizan las acciones de los robots se toma el caso `(accion BloqueaTiroPortero)`. Este comportamiento consiste en que el portero cubra la portería e intercepte la pelota cuando ésta se dirija hacia gol. Para realizarlo es necesario validar si el vector de movimiento de la pelota se intersecta con una línea paralela a la línea de gol a sobre la coordenada *x* del portero. En caso de que la pelota se dirija hacia gol se envía al robot hacia el punto de intersección, utilizando la predicción obtenida por medio del filtro de Kalman [17] en el sistema de visión. Esta predicción permite obtener el vector de movimiento de la pelota y definir si existe o no una intersección con la línea de gol como se muestra en la figura 4.22.

| Comportamiento | Descripción |
|---|---|
| (accion BloqueaTiroPortero) | El portero cubre desde el punto central e intercepta si la pelota se dirige a gol |
| (accion BloqueaTiroPorteroLat) | El portero cubre desde un punto lateral e intercepta la pelota si se dirige a gol |
| (accion DespejarPorterot) | El portero despeja la pelota de su área |
| (accion Despejar) | Despejar la pelota hacia adelante |
| (accion BarreraPorteriaIzq) | Barrera en la portería desde el punto de cobertura izquierdo |
| (accion BarreraPorteriaDer) | Barrera en la portería desde el punto de cobertura derecho |
| (accion RecibePaseAdelanteD1) | Delantero 1 intercepta un despeje |
| (accion RecibePaseAdelanteD2) | Delantero 2 intercepta un despeje |
| (accion RecibePaseAdelanteD3) | Delantero 3 intercepta un despeje |
| (accion MarcarCentralTiroPrimero) | Marcar al contrario mejor posicionado |
| (accion RecibePaseCentroD1) | Delantero 1 intercepta un pase central |
| (accion RecibePaseCentroD2) | Delantero 2 intercepta un pase central |
| (accion PateaGol) | Patear a gol |
| (accion RecibePaseLateralD1) | Delantero 1 intercepta un pase lateral |
| (accion RecibePaseLateralD2) | Delantero 2 intercepta un pase lateral |
| (accion RecibePaseLateralD3) | Delantero 3 intercepta un pase lateral |
| (accion PaseD1)) | Dar un pase a Delantero 1 |
| (accion PaseD2)) | Dar un pase a Delantero 2 |
| (accion PaseD3) | Dar un pase a Delantero 3 |

***Tabla 4.9.*** *Comportamientos para la estrategia de juego.*

La intercepción de la pelota para recibir pases se realiza de forma similar. La diferencia es que el punto de intersección se obtiene a partir de la línea de movimiento de la pelota y una línea trazada entre el robot receptor y un poste superior o inferior de la portería, dependiendo de la ubicación del robot. Lo anterior con la finalidad de que el movimiento del robot se realice en la misma dirección que la dirección de pateo deseada y le sea más fácil meter gol (figura 4.23).

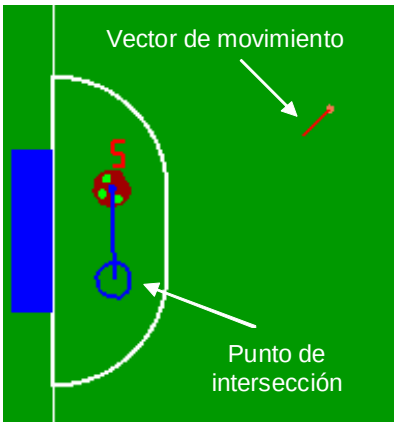



***Figura 4.22.*** *Intercepciónde l portero.*

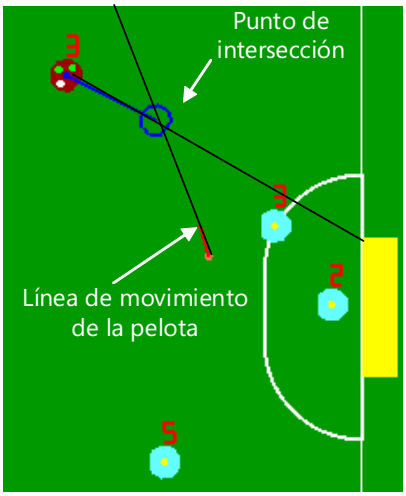



***Figura 4.23.*** *Intercepción de un pase lateral.*

#### 4.8.1.3. Modelado del entorno

Una vez definidas las condiciones y las posibles acciones de los robots se describen las reglas para la estrategia de juego que posteriormente se modelan en el sistema experto. Para ejemplificar la forma en que se modela la estrategia de juego en el sistema experto por medio de reglas se toma el caso de la coordinación entre el portero y dos defensas:

1. En caso de que el portero no cuente con la ayuda de ningún robot en la barrera, entonces realiza el bloqueo desde un punto central de la portería hacia donde se encuentra la pelota.

2. En caso de que cuente con la ayuda de dos robots, el portero seguirá cubriendo desde el punto central y los defensas le ayudaran a cubrir los puntos derecho e izquierdo de la portería.

3. En caso de que sólo cuente con la ayuda de un robot, el portero cubrirá el punto derecho, si la pelota está a la derecha de la mitad de la portería o, el izquierdo, si la pelota está del lado izquierdo.

4. En caso de que la distancia entre la pelota y la portería sea menor a un radio definido, el portero sale a despejar.

Los puntos de cobertura de la portería mencionados se muestran de manera gráfica en la figura 4.24.

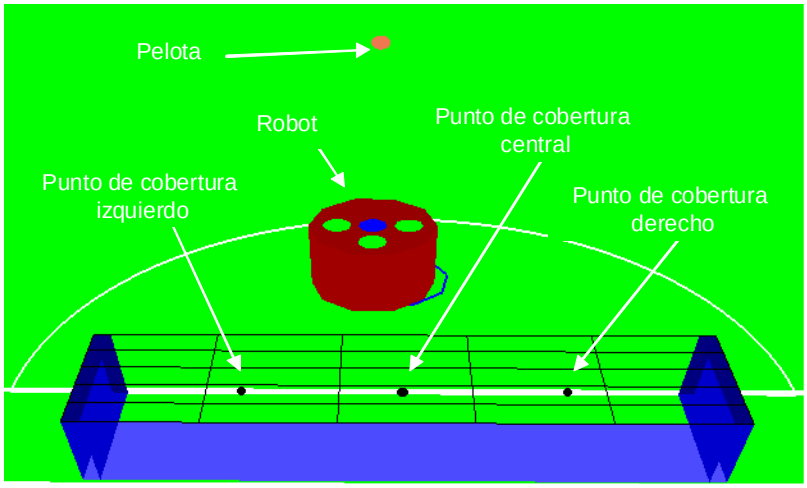



***Figura 4.24.*** *Puntos de cobertura de la portería*

Para modelar este ejemplo se toma únicamente el caso del portero. La tabla 4.10 muestra las reglas diseñadas para el estado de juego START.

De forma similar, se definen las estrategias de los cinco robots para que jueguen en equipo. Las figuras 4.25 a 4.28 a muestran diferentes configuraciones y comportamientos a realizar por cada robot dependiendo de su rol, las condiciones de juego y las reglas definidas en la estrategia para el estado de juego START.

La definición de la estrategia en otros estados de juego se realiza de manera similar al estado START, respetando las transiciones entre los mismos; por ejemplo, al recibir un comando STOP, los robots respetan la regla de mantener una distancia mínima de 500 mm con respecto a la pelota para que el árbitro pueda colocarla y reiniciar el juego. En caso de que el estado de juego sea a favor del equipo contrario, los robots EK respetan la regla de no tocar la pelota hasta que los robots contrarios la pongan en movimiento. Para realizar la transición automática hacia START (figura 4.3) respetando esta regla, se inserta una condición que evalúa el movimiento de la pelota a partir del último punto registrado en el estado STOP. En caso de que el estado de juego sea a nuestro favor, por ejemplo, en un tiro de esquina, las reglas de juego que se aplican al estado de juego OUR_DIRECT_KICK nuestro, colocan a dos robots del equipo EK en posiciones de recepción de pase lateral y pase central. Una vez que llegan a este punto se realiza la transición al estado START, siguiendo la estrategia ofensiva para este estado de juego.

```
 (defrule regla_despeja_portero
      (estado_juego Start)
      (region_pelota DentroAreaPorteriaEK)
=>
      (assert (accion DespejarPortero))
)
(defrule regla_portero_lat         (estado_juego Start)
      (not (region_pelota DentroAreaPorteriaEK))
      (or
      (and
      (or (d3_comp BarreraPorteriaDer) (d3_comp
BarreraPorteriaIzq))
      (or (def_comp BarreraPorteriaDer) (def_comp
BarreraPorteriaIzq)))
      (and
      (not(or (d3_comp BarreraPorteriaDer) (d3_comp
BarreraPorteriaIzq)))
      (not(or (def_comp BarreraPorteriaDer) (def_comp
BarreraPorteriaIzq)))))

=>
      (assert (port_centro))
)
defrule regla_cubre_portero_lat
       (port_centro)
=>
(assert (accion BloqueaTiroPortero))
)
(defrule regla_cubre_portero_centro
      (estado_juego Start)
      (not (region_pelota DentroAreaPorteriaEK))
      (not (port_centro))
=>
      (assert (accion BloqueaTiroPorteroLat))
)
```

***Tabla 4.10.*** *Reglas para el juego del portero.*

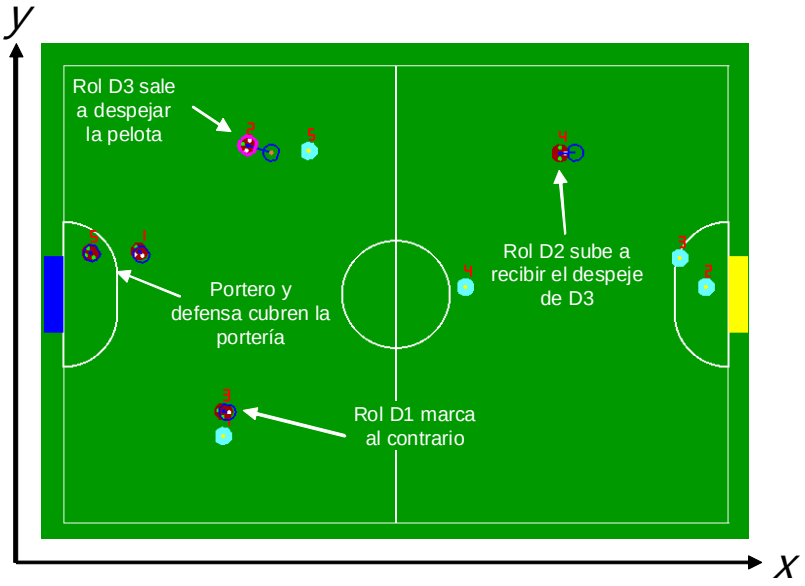



***Figura 4.25.*** *Primer ejemplo de estrategia de juego.*

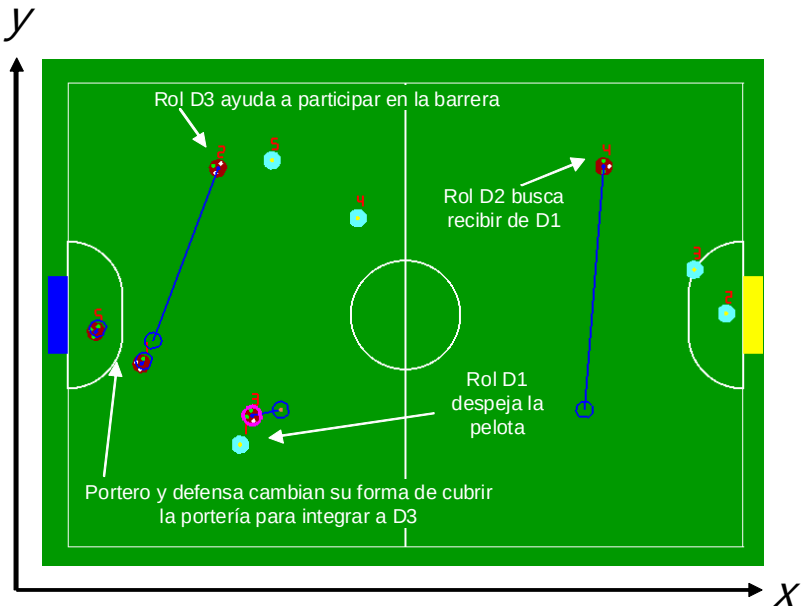



***Figura 4.26.*** *Segundo ejemplo de estrategia de juego.*

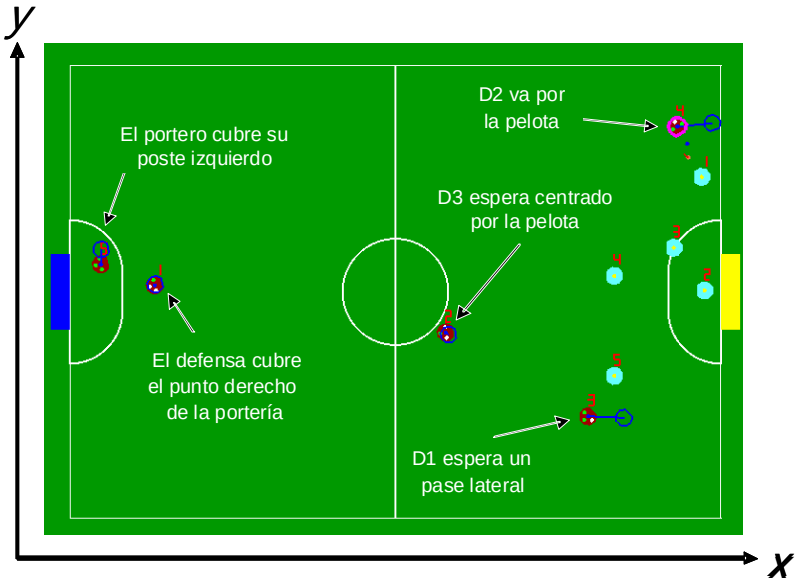



***Figura 4.27.*** *Tercer ejemplo de la estrategia de juego.*

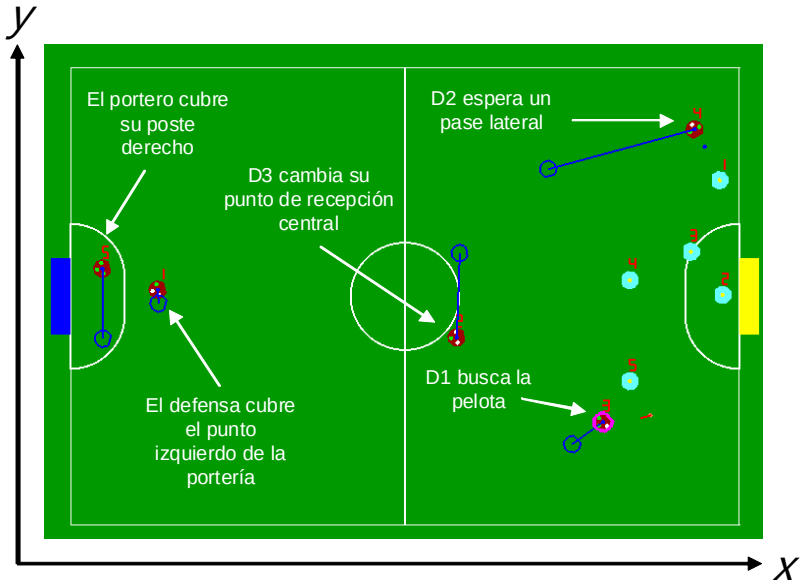



***Figura 4.28.*** *Cuarto ejemplo de la estrategia de juego.*

### 4.8.2. Ejecución de comportamientos

Una vez que le compolrtamiento ha sido definido por el sistema experto, es necesario llevar a cabo la ejecución de dicha acción. En cada ciclo de control se recibe la información actualizada del sistema de visión que nos permite actualizar la coordenada y orientación a las que deseamos se dirija el robot. La ejecución de cualquier acción consiste en definir lo siguiente:

- Coordenada final.
- Orientación final.
- Velocidad lineal máxima.
- Velocidad de giro máxima.

Tomando como ejemplo el comportamiento de seguir a la pelota, el movimiento del robot puede resultar en una línea recta, si la pelota está detenida (figura 4.29), o en otra trayectoria, si la pelota está en movimiento (figura 4.30). Esta forma de ejecutar una acción es útil para la gran mayoría de los comportamientos como marcar, cubrir la portería, cubrir la pelota, etc.

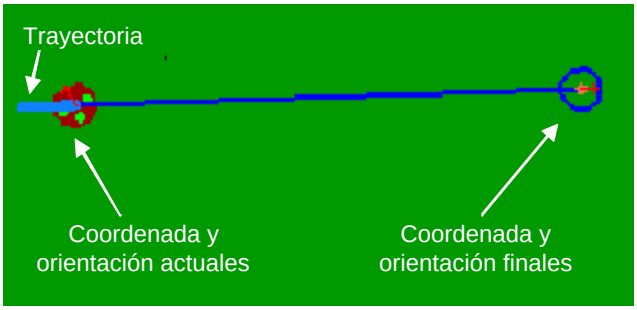



***Figura 4.29.*** *Trayectoria hacia la pelota estática.*

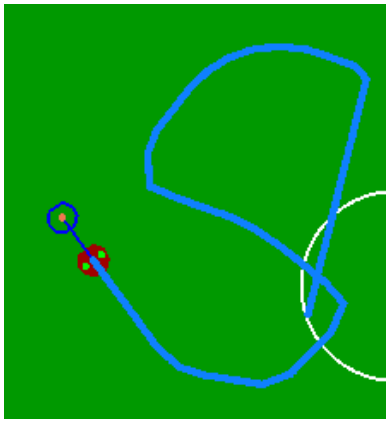


***Figura 4.30.*** *Trayectoria hacia la pelota en movimiento.*

Existen otras acciones que requieren definir una trayectoria diferente a una línea recta entre una posición inicial y una final, por ejemplo:

1. Realizar un *zig-zag*, en donde el robot debe pasar por una serie de puntos a lo largo de la cancha.

2. Aproximar al robot hacia la pelota en dirección a un punto de tiro que puede ser la portería contraria u otro robot, en donde el robot debe pasar por una serie de puntos antes de llegar a la pelota.

Una solución es definir un punto intermedio para que cuando el robot llegue, cambie su meta hacia la coordenada final. Existen varios problemas para determinar si el robot ha llegado al punto intermedio, ya que es necesario definir un radio de aproximación de la misma forma que se hace para saber si el robot ha llegado a su coordenada final. La principal diferencia de definir un radio de aproximación alrededor de un punto intermedio y alrededor de la coordenada final es que es que el módulo de control frena al robot antes de llegar a la coordenada final, permitiendo que este radio sea lo suficientemente pequeño como para no perjudicar la precisión de llegada. Lo anterior no sucede cuando se define un radio de aproximación en un punto intermedio, ya que la velocidad cerca de este punto es mayor y no deseamos frenar al robot. El radio de aproximación requerido para determinar si el robot llegó a dicho punto tiene que aumentar, ocasionando que el robot se adelante y se dirija al punto final antes de llegar al punto intermedio.

Otro problema de definir un punto intermedio es que cuando se realiza un cambio drástico de dirección entre un ciclo de control y otro, al robot se le dificulta seguir la

trayectoria deseada debido al tiempo de la respuesta del controlador PID y al modelo de aceleración y desaceleración. Una solución sin recurrir al uso de puntos intermedios es realizar la interpolación de los puntos inicial, final e intermedios por medio de un *spline.*

## 4.9. Control de trayectorias

El control de trayectorias que siguen los robots consta de dos etapas y se realiza después de la ejecución del comportamiento. La primera es la generación de trayectorias a partir de una serie de puntos por los cuales se desea que navegue el robot de forma suave y se basa en *splines*. La segunda es la planeación de rutas para la evasión de obstáculos por medio de árboles GET. Las dos metodologías se integran para posteriormente realizar el control de movimientos.

### 4.9.1. Trayectorias basadas en splines

Para demostrar la generación de trayectorias a partir de una serie de puntos intermedios, se tomará el ejemplo del robot aproximándose a la pelota para tirar hacia la portería contraria. Suponiendo que la pelota se encuentre estática en la cancha, se definen los puntos de control con los que se genera una trayectoria que forma parte de una curva que satisface las condiciones de un *spline* cúbico. Existen dos posibilidades: que la pelota se encuentre delante o detrás del robot, tomando como referencia su eje *x* (figura 4.15).

En caso de que la pelota se encuentre adelante del robot, se definen cuatro puntos de control: el inicial, dos de aproximación detrás de la pelota en dirección a la portería contraria y el de la pelota.

La figura 4.31 describe la evaluación de 23 puntos del *spline* generados a partir de los cuatro puntos de control. Se utilizaron dos puntos de aproximación detrás de la pelota, uno a 100 mm y otro a 300 mm en dirección a la portería contraria. El resultado del *spline* es una serie de puntos que definen la trayectoria ideal por los que debería pasar el robot para llegar correctamente a la pelota. Para definir la coordenada final en cada ciclo de control, se mantiene una distancia igual a la longitud de la trayectoria en la dirección que hay entre el primer punto del *spline* (coordenada del robot) y el segundo punto. La finalidad de mantener la distancia es evitar que el módulo de control frene al robot antes de que llegue a su posición final. La orientación final se calcula como el ángulo entre el robot y el segundo punto del *spline*.

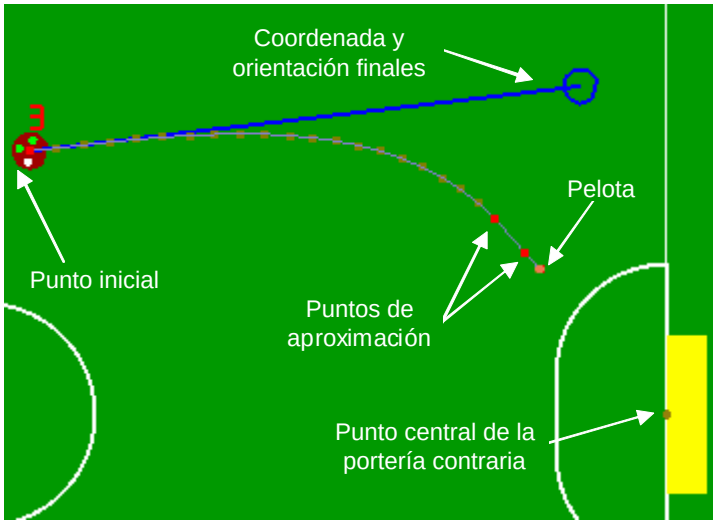



***Figura 4.31.*** *Trayectoria hacia la pelota en dirección a la portería.*

En caso de que la pelota se encuentre detrás robot se agrega un punto de control adicional que puede estar del lado derecho o izquierdo, dependiendo de su ubicación respecto al robot (figura 4.32).

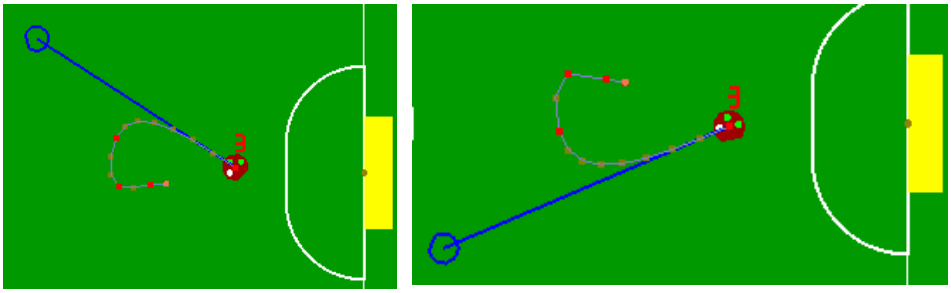


***Figura 4.32.*** *Aproximación desde atrás por ambos lados de la pelota.*

Conforme el robot se va moviendo en cada ciclo de control, se genera un nuevo *spline* y se actualiza la trayectoria, ya que la pelota puede estar en movimiento.

Para que la navegación funcione correctamente es necesario resolver lo siguiente: en caso de que el robot se acerque al siguiente punto de control es necesario eliminarlo una vez que éste se ubique detrás del robot para evitar que el robot retroceda sobre la trayectoria del *spline*. Se debe evitar medir la distancia que hay entre este punto de control y el robot, ya que sería equivalente a utilizar un punto intermedio. Una solución para determinar el momento en el que el robot sobrepasa el punto de control es llevar un registro de la última orientación obtenida para detectar cuándo existe una variación muy grande. Esto sólo ocurre cuando el robot sobrepasa el punto de control en dirección hacia él por lo que se elimina este último y se genera nuevamente el *spline*.

Otro caso se presenta cuando el robot está muy cercano a la pelota y resulta más conveniente enviarlo directamente hacia ella y dejar de generar el *spline*.

La siguiente secuencia de imágenes de la figura 4.33, muestra la forma en la que un robot sigue el *spline* hasta su meta.

La trayectoria completa es descompuesta en distintas coordenadas finales que al igual que una trayectoria lineal requieren de algún método de planeación de rutas para evitar que el robot colisione con algún obstáculo.

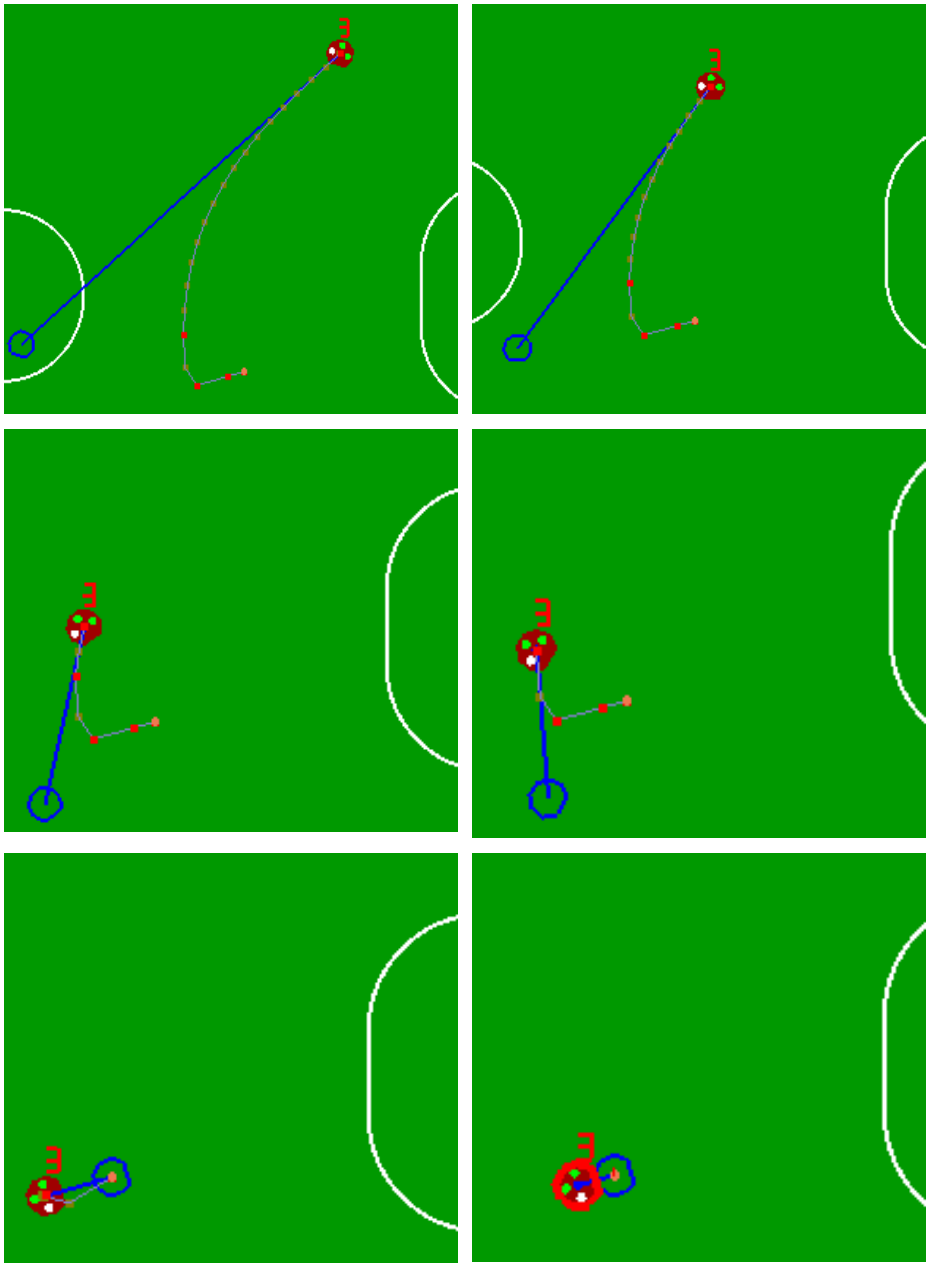


***Figura 4.33.*** *Aproximación a la pelota por medio de un spline.*

### 4.9.2. Planeación de rutas por medio de árboles GET.

Para el problema de la evasión de obstáculos se proponen los árboles de exploración geométrica (GET) como variante de los RRT. Este algoritmo busca aprovechar el conocimiento que se tiene acerca de cómo pueden estar configurados el espacio y la forma que tienen los obstáculos y así eliminar la aleatoriedad del algoritmo.

En el campo de juego, los obstáculos pueden representarse como figuras geométricas para que el planificador pueda utilizar la forma de estos objetos y así encontrar una ruta eficiente entre el punto de inicio y la meta. El árbol de exploración geométrica es una forma eficiente de construir un árbol en cada ciclo de control y también es capaz de combinar diferentes tipos de obstáculos. En nuestro problema, los robots son cilíndricos y se identifican por medio del sistema de visión global en dos dimensiones por lo que pueden representarse por medio de círculos de 180 mm de diámetro. Otros obstáculos como las porterías, pueden representarse como rectángulos.

Las acciones de la estrategia de juego definen una posición hacia la cual deseamos mover al robot. Para determinar la ruta más apropiada, partimos de la definición de un punto inicial y un punto final. El punto inicial lo consideramos como el primer y, por el momento, único nodo de exploración, y como raíz del árbol.

La figura 4.34 muestra un escenario para la planeación de una ruta hacia la pelota con un solo obstáculo obstruyendo el movimiento en línea recta.

***Figura 4.34.*** *Planeación de una ruta hacia la pelota.*

Para realizar el proceso se crea una lista circular dentro de la cual se tienen los apuntadores a los nodos del árbol que sirven para ir realizando el proceso de exploración. Cada nodo del árbol está formado por una coordenada. En cada iteración se evalúa un nodo de esta lista y al finalizar la evaluación se selecciona el siguiente nodo de la lista. El algoritmo termina cuando algún nodo de exploración alcanza la meta.

Un nodo de exploración puede tener dos estados: El primero ocurre cuando está rodeando algún obstáculo de forma tangencial y el segundo cuándo se está dirigiendo en línea recta hacia la meta. En caso de que el nodo de exploración se encuentre rodeando algún obstáculo, se asigna una referencia a dicho obstáculo así como el sentido en el que se rodea dicho obstáculo. El diagrama de flujo del algoritmo se muestra en las figuras 4.36 y 4.37.

Para determinar si existe un obstáculo interfiriendo para la extensión del nuevo punto, se generan los puntos de intersección entre el nodo de exploración y la meta (figura 4.35).

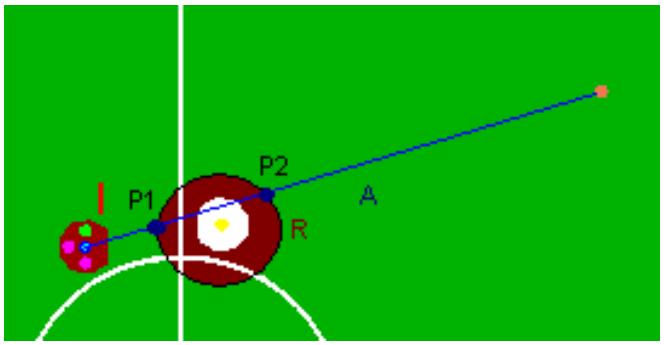



*Figura 4.35. Intersección de un obstáculo con el punto de exploración inicial.*

La intersección se obtiene a partir de la ecuación del círculo con centro en el obstáculo a evaluar y de la ecuación de la recta formada por los puntos de inicio y fin como se muestra a continuación:

*Ax, Ay:* coordenada del robot.
*Bx, By:* coordenada del obstáculo que se está evaluando.
R: radio de evasión.
*Cx, Cy:* coordenada final.
*P1x, P1y:* primer punto de intersección.
*P2x, P2y:* segundo punto de intersección.

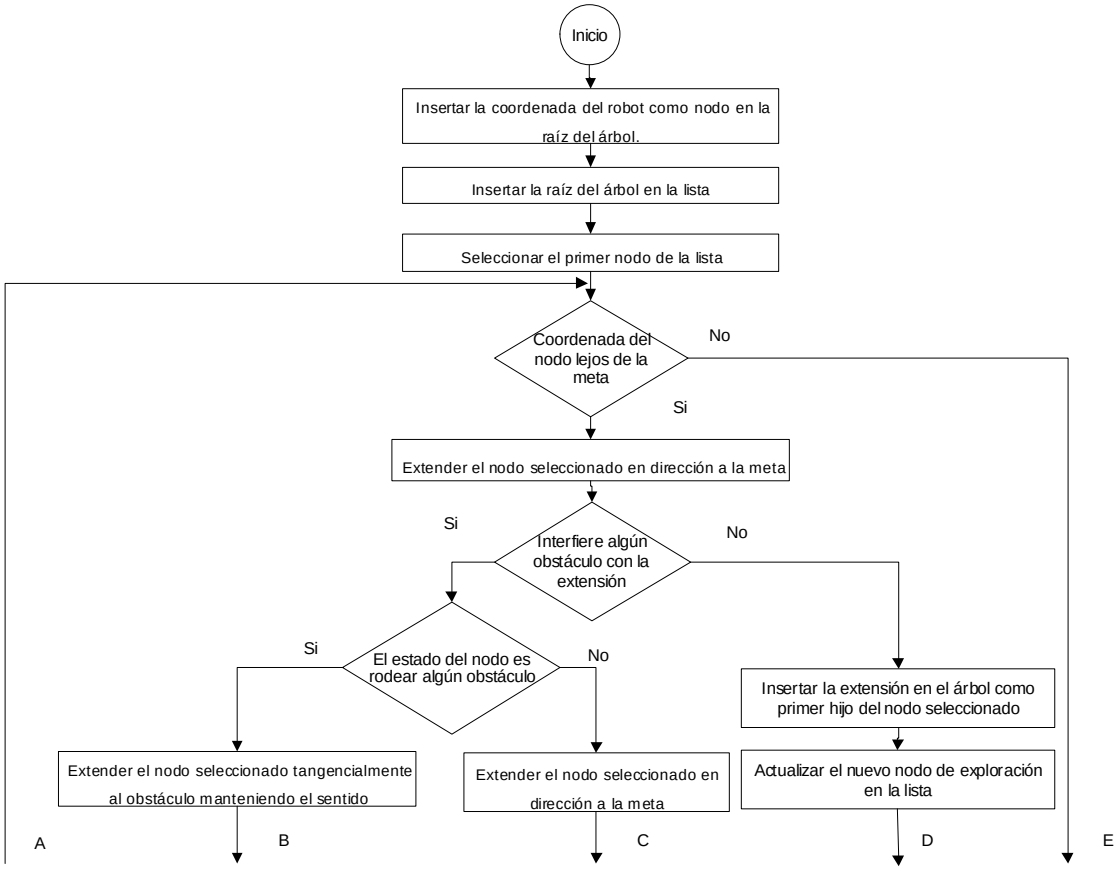



***Figura 4.36.*** *Algoritmo GET (primera parte).*

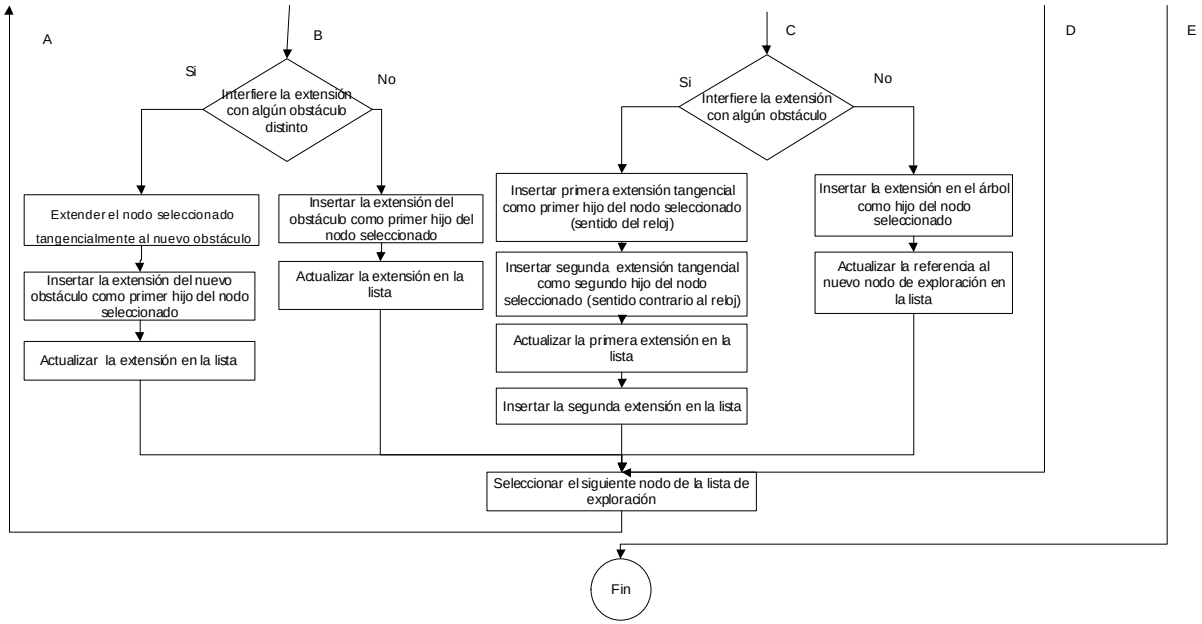



***Figura 4.37.*** *Algoritmo GET (segunda parte).*

Intersectando la ecuación de la recta con la ecuación del círculo se obtienen los coeficientes (a, b, c) de la ecuación de segundo grado resultante, validando que exista la ecuación de la recta:

Sea la ecuación de la recta:

$$\begin{aligned} &\text{Si } Cx - Ax \neq 0 \\ &m = (Cy - Ay)/(Cx - Ax) \\ &y = m(x - Ax) + Ay \end{aligned} \qquad (4.17)$$

sustituyendo 4.17 en 4.18:

$$y = (Cy - Ay)/(Cx - Ax)(x - Ax) + Ay \qquad (4.18)$$

Y la ecuación del círculo:

$$R^2 = (x - Bx)^2 + (y - By)^2 \qquad (4.19)$$

Sustituyendo la ecuación 4.18 en 4.19 se obtiene la siguiente ecuación de segundo grado:

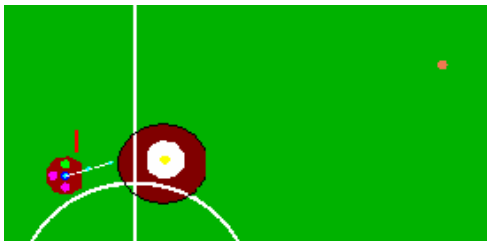
$$\begin{aligned} &ax^2 + bx + c = 0 \\ &a = 1 + m^2 \\ &b = 2mAy - 2Bx - 2m^2Ax - 2mBy \\ &c = Bx^2 + m^2Ax^2 + Ay^2 + By^2 - 2AyBy - 2mAxAy + 2mAxBy - R^2 \end{aligned} \qquad (4.20)$$

Finalmente resolviendo la ecuación 4.20 se obtienen los puntos de intersección si estos existen:

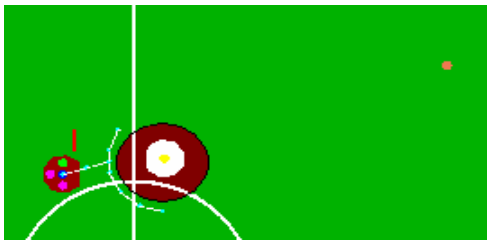
$$\text{Si } b^2 >= 4ac$$

$$\begin{aligned} &P1x = (-b + \sqrt{(b^2 - 4ac)})/2a \\ &P1y = m\,(P1x - Ax) + Ay \\ &P2x = (-b - \sqrt{(b^2 - 4ac)})/2a \\ &P2y = m\,(P2x - Ax) + Ay \end{aligned} \qquad (4.21)$$

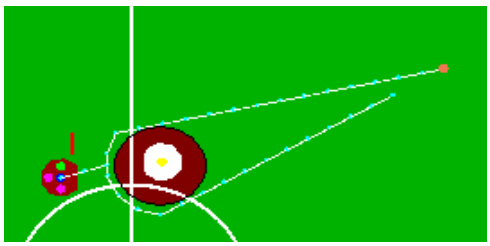
En caso de que no existan puntos de intersección y la distancia de la siguiente extensión sea menor al radio de evasión, se omite el obstáculo. Si se cumplen estas condiciones y además se encuentran dos puntos de intersección, se toma como referencia el punto más cercano al robot. El siguiente paso es determinar si la extensión se ubica dentro de un radio por lo menos del doble del radio del robot. Las figura 4.38 ilustra el crecimiento del árbol y la forma en que se rodea al obstáculo.



a) El nodo de exploración inicial avanza hacia la meta.



b) Se generan dos nodos de exploración y cada uno rodea al obstáculo en un sentido.



c) Los nodos de exploración terminan de rodear y se dirigen hacia la meta.

***Figura 4.38.*** *Algoritmo GET.*

La generación del árbol se realiza en cada ciclo de control. Conforme el robot avanza se acerca al punto en donde se generaron dos posibles rutas. Suponiendo que la ruta se mantiene constante durante todo el recorrido, el robot toma la ruta que primero llegó al destino (figura 4.39).

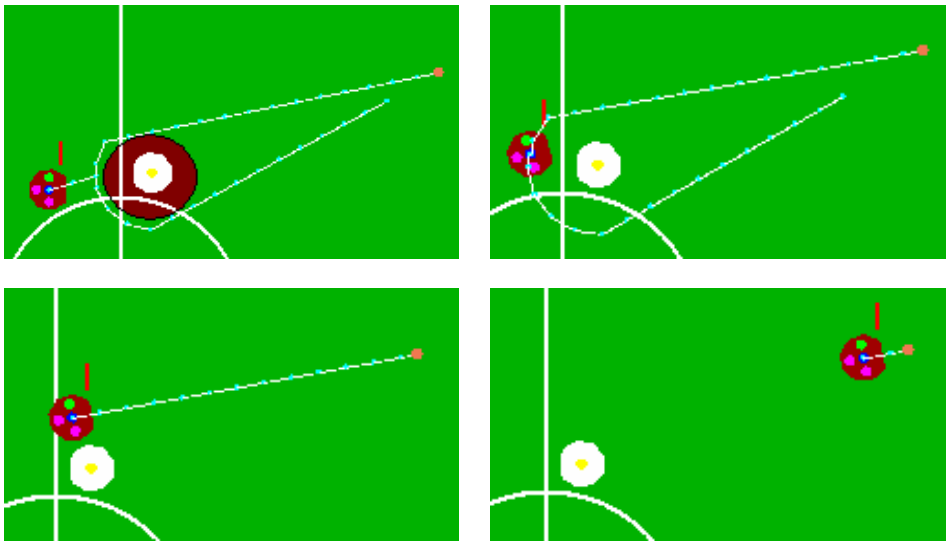


***Figura 4.39.*** *Movimiento del robot a lo largo de la trayectoria.*

Es posible que las dos rutas lleguen al destino al mismo tiempo, o bien, que durante el seguimiento de una ruta se decida cambiar, lo que generaría un estado de inestabilidad. Este problema se presenta una vez que el robot está rodeando al obstáculo, por lo que para solucionarlo se mantiene el mismo sentido de rodeo entre dos ciclos de control. Este sentido se reinicia cuando se encuentra un nuevo obstáculo o cuando se puede dirigir en línea recta hacia la meta. La generación del árbol cuando existe más de un obstáculo se puede ver en la figura 4.40.

Como se observa en las imágenes, los robots contrarios o los nuestros se modelan como obstáculos circulares. En el caso de una portería es posible simplificar la generación del árbol para mantener un nodo de exploración y considerarla como un rectángulo. Se puede observar cómo existen varias rutas que pasan a diferentes distancias de los obstáculos. Esto se debe a un pequeño error introducido por el tamaño de las extensiones, sin embargo, es un rango muy pequeño que no afecta el desempeño del algoritmo.

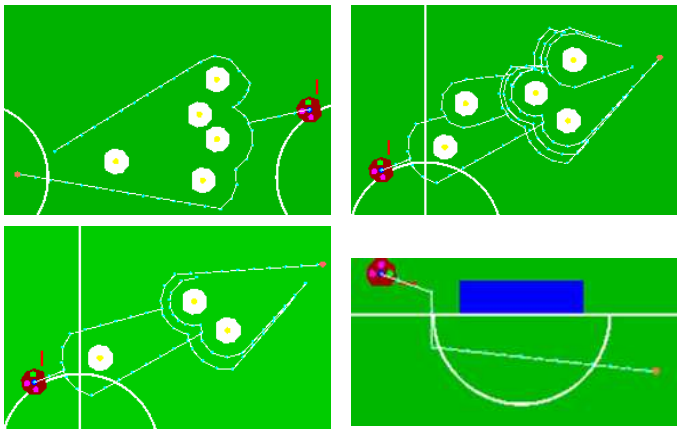


***Figura 4.40.*** *Generación del árbol bajo diferentes configuraciones.*

### 4.9.3. Integración para el control de trayectorias

En caso de que un robot esté siguiendo la trayectoria de un *spline* y se presenten uno o más obstáculos, se genera el árbol hacia el punto final definido por el *spline* que varía en cada ciclo de control y no hacia la meta final.

La figura 4.41 ejemplifica cómo se selecciona la coordenada final a partir de evaluar el *spline* y, posteriormente, la generación del árbol hacia esta coordenada.

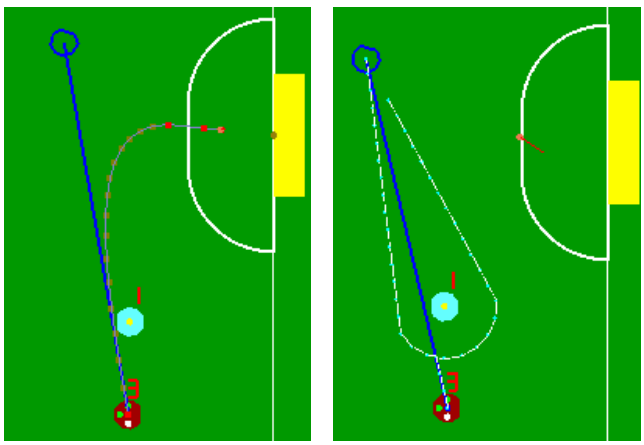


***Figura 4.41.*** *Generación del árbol al seguir la trayectoria de un spline.*

# Capítulo 5

## IMPLEMENTACIÓN

La implementación del sistema de IA parte del *hardware* y *software* utilizados para su desarrollo. Posteriormente se describe la implementación de cada una de las funcionalidades, tomando como referencia las entradas y salidas del sistema de IA. El filtro de Kalman y los controladores PID se implementaron en el sistema de visión y en los robots respectivamente. Se incluyeron en el análisis del presente trabajo por ser indispensables para comprender la arquitectura del sistema de IA.

### 5.1. Herramientas y *software*

El sistema fue probado en una *laptop* IBM con procesador Intel Pentium 4 a 2 GHz y Memoria RAM de 512 MB. Para el desarrollo del sistema se utilizó Microsoft Visual C++ 6.0 con MFC *(Microsoft Foundation Class Library)* para desarrollar la interfaz gráfica y ha sido probado sobre los sistemas operativos Windows 2000 y Windows XP. Se utlizaron librerías externas requeridas para la comunicación con el puerto paralelo, el *joystick* por medio de las librerías de *direct-input*, la librería de CLIPS para el sistema experto, las librerías de OpenGL para el ambiente gráfico y la generación de splines cúbicos.

La interfaz gráfica se compone de un diálogo principal que contiene una serie de diálogos desplegados dentro de un tabulador del lado izquierdo, así como otro diálogo sobre el cual se dibuja el ambiente gráfico (figura 5.1)


***Figura 5.1.*** *Interfaz gráfica del sistema EKIntelSSL.*

El sistema funciona con varios *threads* (hilos de ejecución). El *thread* principal realiza todo el ciclo de procesamiento de la aplicación. En paralelo con este *thread* se puede interactuar con la interfaz gráfica de los diferentes diálogos. En el caso del simulador y el ambiente gráfico, se utilizan dos *timers* diferentes, cada uno en su diálogo correspondiente. Un segundo *thread* se utiliza para recibir la información del *referee box*. De esta forma, la aplicación está trabajando sobre los objetos contenidos en el manejador principal que pueden ser modificados o accedidos por distintas partes de la aplicación.

Dentro del ciclo principal se realizan las siguientes funciones:

1. Recepción del sistema de visión.
2. Actualización de información en las interfaces del usuario.
3. Evaluación de la estrategia de juego.
4. Evaluación del control de trayectorias.
5. Lectura de los eventos del *joystick*.
6. Control de trayectorias.
7. Control de movimientos.
8. Envío de información.

## 5.2. Arquitectura del sistema de IA

El sistema de IA utiliza diversas conexiones para formar la arquitectura completa. La figura 5.2 muestra de que forma interactúa el sistema con sus entradas y salidas.

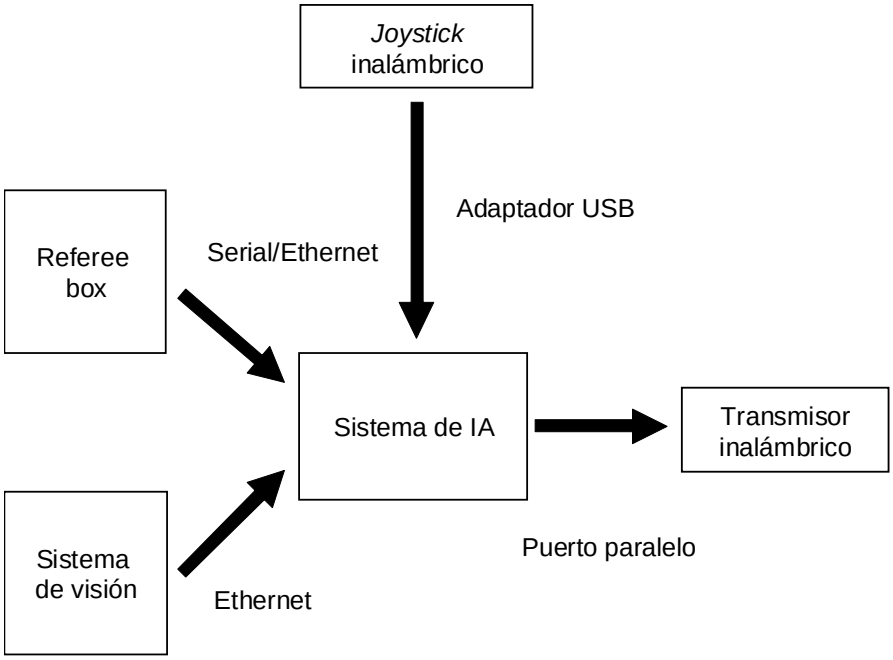



***Figura 5.2.*** *Entradas y salidas del Sistema de IA.*

El usuario realiza la configuración de las comunicaciones del sistema como se muestra en la figura 5.3.

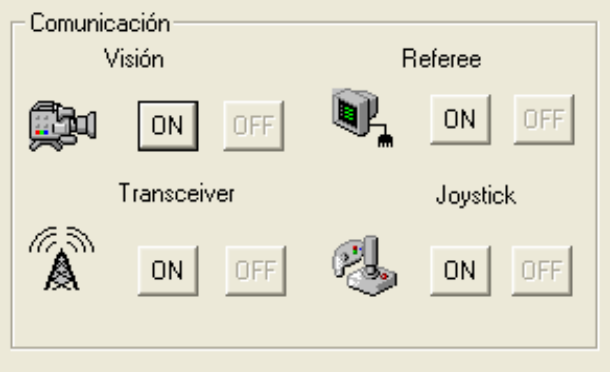



***Figura 5.3.*** *Control de entradas y salidas.*

### 5.3. Recepción del sistema de visión

El sistema de visión está compuesto por dos cámaras *Guppy G-036C* de la marca *AVT* (*Allied Vision Technology*) y por elementos de conexión que sirven para obtener las señales de video en una computadora [29] La figura 5.4 muestra las entradas (señales digitales de video multiplexadas por un *Hub Firewire* de cada una de las cámaras) y las salidas (información acerca de la localización de los robots y la pelota dentro del campo de juego) del sistema de visión hacia el sistema de IA.


***Figura 5.4.*** *Entradas y salidas del sistema de visión hacia el sistema de IA.*

La comunicación entre el sistema de visión y el sistema de IA se realiza por medio de una red local. La tabla 5.1 muestra la configuración de las direcciones IP de cada una de las computadoras.

La dirección IP y la máscara de red deben configurarse seleccionando el puerto *ethernet* que se va a utilizar en las `Conexiones de red` y posteriormente en propiedades de Protocolo Internet (TCP/IP) (figura 5.5).

| Computadora | Dir IP | Máscara de Red | Socket |
|---|---|---|---|
| EKVision | 192.168.0.2 | 255.255.255.0 | 5001 |
| EKIntelSSL | 192.168.0.1 | 255.255.255.0 | 5001 |

***Tabla 5.1.*** *Comunicación entre sistemas de visión y de IA.*

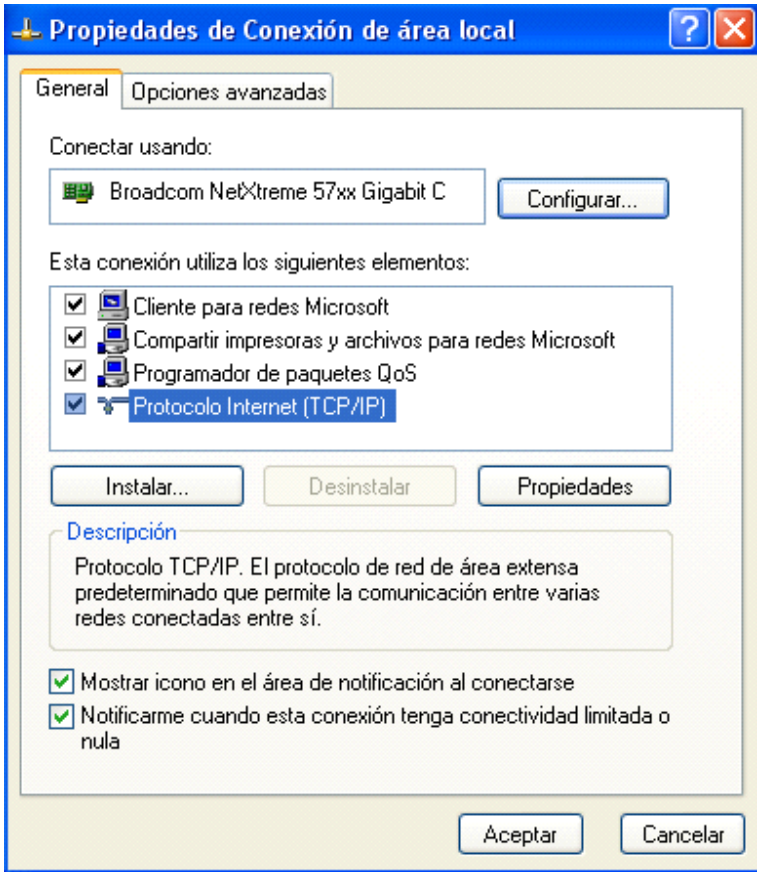



***Figura 5.5.*** *Configuración de la dirección IP y la máscara de Red.*

## 5.4. Recepción e interfaz del referee box

La conexión con el *referee box* se realiza por medio de una conexión LAN por *ethernet*. Debido a que la computadora sólo tiene un puerto *ethernet* que se ocupa en la interconexión con el sistema de visión, se utiliza el convertidor *USB-Ethernet* de la figura 5.6.

*Figura 5.6. Convertidor USB-Ethernet.*

La configuración de la dirección IP y el *Socket* UDP se puede encontrar en la página dedicada al *referee box* en el sitio oficial de la liga *Small Size*. [31].

El usuario configura el color del equipo EK y el sentido en el que patea el equipo como se muestra en la figura 5.7.

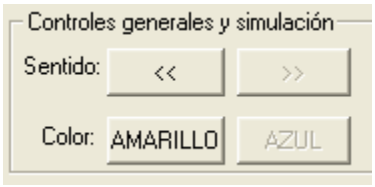



*Figura 5.7. Configuración del color y el sentido de juego*

La interfaz del referee permite al usuario generar estos comandos dentro del sistema de IA de forma similar a como se realiza durante un partido oficial (figura 5.8).

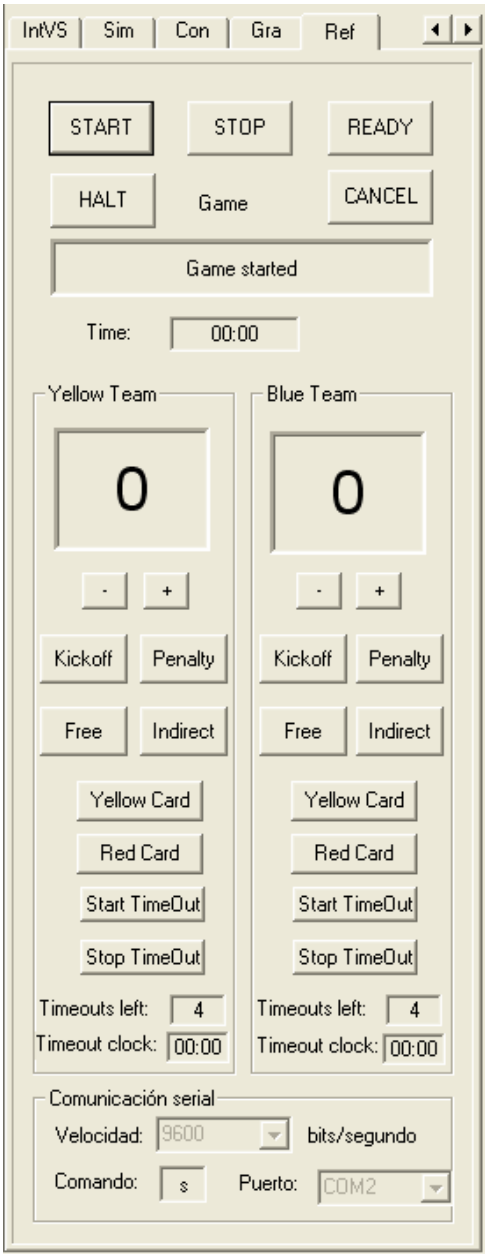



*Figura 5.8. Interfaz del referee.*

## 5.5. Visualización y ambiente de pruebas

El ambiente de pruebas se implementó de la siguiente forma: el controlador externo se conecta directamente a la computadora para el control del robot. La interfaz de control cuenta con dos formas de manipular a los robots: por el ratón de la computadora y por la definición de trayectorias.

### 5.5.1. Controlador externo (joystick)

El sistema EKIntelSSL utiliza un *joystick* inalámbrico para probar el control de los robots. Es posible conectar cualquier otro dispositivo que pueda detectarse por medio de las librerías *Direct Input* de Microsoft, pero la respuesta a la sensibilidad del *joystick* para el movimiento omnidireccional y la numeración de los botones para el uso de los dispositivos del robot, están programadas para el modelo inalámbrico *Logitech® Freedom™ 2.4 Cordless Joystick* que se conecta por un puerto USB (figura 5.9).

***Figura 5.9.*** Joystick *inalámbrico.*

La figura 5.10 muestra la forma en que el usuario configura el número de robot que se desea controlar y el equipo al que pertenece (el número de robot se puede modificar por medio del *slider* del *joystick*).


***Figura 5.10.*** *Configuración del joysitick.*

Los parámetros utilizados para manejar a los robots por medio del *joystick* son los siguientes:

`ang_sum_joy:` ángulo que se suma al ángulo actual para definir la orientación final por medio del *joystick*.

`division_ejes:` número de niveles de velocidad que puede tener cada eje del *joystick*.

`rango_giro:` rango de sensibilidad para anular el movimiento de giro del *joystick*.

`sensibilidad:` rango de sensibilidad para anular el movimiento del *joystick*.

### 5.5.2. Interfaz de control de pruebas

La interfaz de control (figura 5.11) permite reali-zar pruebas con los robots por medio de dos métodos de prueba: `Ratón` o `Línea Recta.`


***Figura 5.11.*** *Interfaz de control.*

En el método de Ratón, se puede seleccionar cualquier punto de la cancha para definir la coordenada a la que se desea mover el robot sin modificar su orientación.

En el de Línea Recta, se define el vector de movimiento de forma polar o cartesiana. Cuando se activa el *checkbox* Polar, se escribe la distancia y el ángulo en el que se quiere mover al robot, así como la orientación final. En caso de que no estar activo, se escribe cualquier coordenada del campo de juego.

Para iniciar el movimiento o detenerlo se utilizan los botones Iniciar y Detener en ambos métodos de prueba. Es posible tam-bién probar el funcionamiento del *dribbler* y del *kicker* por medio de los controles Dispositivos del Robot. El *dribbler* se controla por medio del *checkbox* Dribbler y el *kicker* se controla con siete botones correspondientes a los niveles de pateo. Dentro de los métodos de prueba se puede activar o desactivar la evasión de obstáculos con el *checkbox* Evasión. El valor de la velocidad del *slider* y el modo de prueba se mantienen en el archivo ControlMovimientoInicializacion.txt.

### 5.5.3. Interfaz de los robots

La interfaz de los robots (figura 5.12) permite controlar la configuración de los robots del equipo EK y a los del equipo contrario. El control de cada uno de los robots consta de tres columnas. En la primera se muestra su número, una imagen de la cubierta que identifica el equipo al que pertenece (cubierta negra significa que es nuestro y cubierta blanca que es contrario) y el código de colores utilizado para la localización en el sistema de visión, el *checkbox* Activado (activa o desactiva para sólo tomarlo en cuenta cuando está encendido y en juego) y el *checkbox* Trama # (decide si la trama correspondiente a su número será la del robot nuestro o la del robot contrario) que sólo se encuentra activado para los robots del primer dialogo. En la segunda columna se cuenta con un *combo-box* que define el rol que va a tomar el robot para la estrategia de juego y con el recuadro Ubicación (indica si en cada iteración, la posición actual del robot se obtiene por medio del sistema de visión o por el simulador). En la tercera columna se cuenta con el cuadro Trayectoria (define la forma de control del robot por medio de la toma de decisiones, el *joystick* o la interfaz de control).

Aunque en un partido se controlan únicamente a los cinco robots de un equipo, es muy útil poder manipular a los robots de ambos equipos en el momento de programar los comportamientos de cada uno de ellos, por ejemplo, 2 contra 3 ó 2 contra 2. Además, el sistema puede controlar hasta 10 robots jugando 5 contra 5, ejecutando el sistema con dos computadoras y utilizando canales de frecuencia distintos para cada equipo.

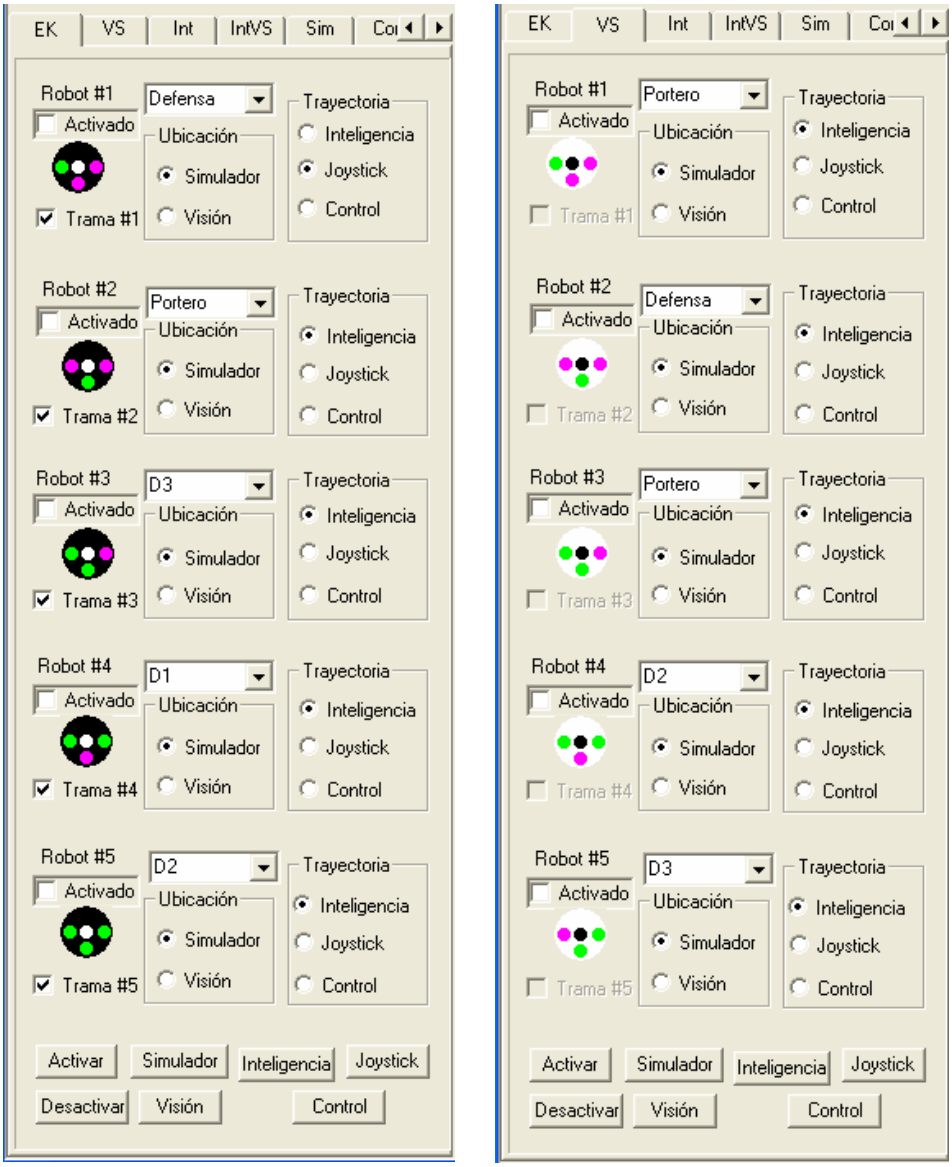



***Figura 5.12.*** *Interfaz de los robots.*

### 5.5.4. Visualización en dos y tres dimensiones

Las configuraciones relacionadas con el ambiente gráfico se realizan por medio de la ventana mostrada en la figura 5.13.

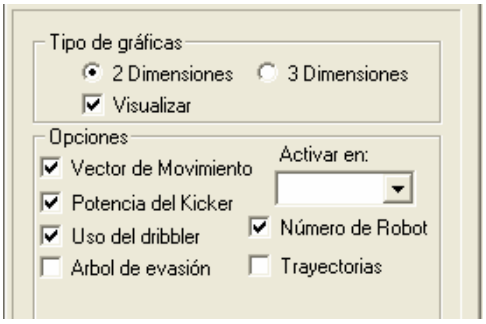



***Figura 5.13.*** *Interfaz del ambiente gráfico.*

El cuadro `Información` proporciona datos de los diez robots y la pelota; muestra para cada robot la posición actual, la posición futura, el nivel de pateo, el uso del *dribbler*, la velocidad de movimiento (descompuesta en cada eje), la velocidad de giro, la velocidad de movimiento lineal y la velocidad medida por el sistema de visión, así como la posición de la pelota (figura 5.14).

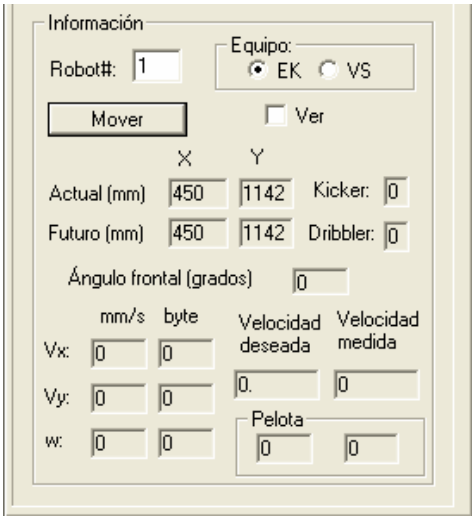



***Figura 5.14.*** *Información de los robots y la pelota.*

El control de la cámara en el modo de tres dimensiones se realiza seleccionando la cancha y utilizando las siguientes teclas:

`Flecha arriba:` avanzar hacia delante.
`Flecha abajo:` avanzar hacia atrás.
`Flecha derecha:` girar a la derecha.
`Flecha izquierda:` girar a la izquierda.
`Re Pag:` girar hacia abajo.
`Av Pag:` girar hacia arriba.

## 5.6. Control de movimientos

La implementación del control de movimiento se distribuye de la siguiente forma: el envío de información describe la forma en que se realiza la comunicación inalámbrica entre el sistema de IA y los robots, los controladores PID se implementan en los robots y el control autónomo se realiza en el sistema de IA por utilizar información del sistema de visión.

### 5.6.1 Envío de información

La conexión con el transmisor inalámbrico se realiza por medio del puerto paralelo, utilizando un dispositivo de *Radiometrix* que consta de un radio transmisor/receptor de UHF y un controlador de paquetes de datos de 40Kbit/s.

El RPC *(Radio Packet Controller)* es un sistema de comunicación que requiere una antena, una fuente de poder de 5V y algún microcontrolador con un puerto bidireccional de 1 byte. Un paquete de datos de 1 a 27 bytes es recibido de un microcontrolador y se aloja en el *buffer* de datos del RPC. El paquete es transmitido por el *transciever* del RPC para ser

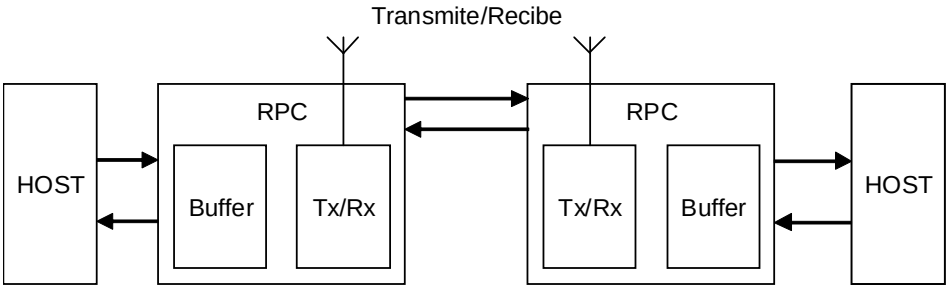



***Figura 5.15.*** *Comunicación por medio del RPC con los robots*

recibido en el *buffer* de datos del RPC destino (figura 5.15) [14].

La figura 5.16 muestra la base utilizada para realizar la conexión entre el sistema de IA y el RPC por medio de una tarjeta electrónica utilizando el puerto paralelo.

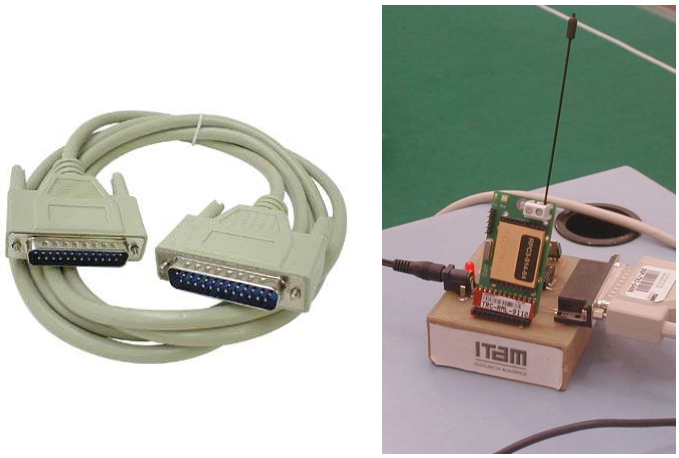


***Figura 5.16.*** *Conexión entre la computadora y el RPC.*

### 5.6.2 Controladores PID

El modelo de control omnidireccional por medio de controladores PID se implementa dentro del DSP de los robots ya que es necesario que interactúe los las señales de lectura y escritura de los motores.

### 5.6.3 Control autónomo

Existen dos formas de ajustar las tres velocidades utilizadas en el simulador: máxima, de giro y de pateo. La primera es la generación `Personalizada`, en donde cada uno de los valores se ingresa directamente en el sistema. La generación `Automática` permite ajustar las tres velocidades utilizando el *slider* de `Velocidad control` que se encuentra en el diálogo de control. De esta forma, la relación entre velocidad real (mm por segundo) y el porcentaje definido por el *slider* se realiza por los parámetros de simulación `vel_50%` y `giro_50%` como se muestra al final de la ventana `(Velocidad del drive control (50%)` y `Velocidad real 100%).`

La ventana de Control tiene dos finalidades. En primer lugar permite configurar los parámetros que son necesarios para controlar a los robots de forma autónoma. Estos parámetros son utilizados para definir la velocidad del robot y el valor de velocidad de los vectores de movimiento que se envía por el *transceiver*. En segundo lugar permite ajustar ciertos parámetros utilizados cuando se desea controlar a los robots por medio del *joystick*.

A continuación se explican los parámetros utilizados del recuadro `Parámetros de control de movimiento:`

`err_dist_drive:` margen para decidir si un robot ha llegado a su posición final (considerando el error de medición del sistema de visión).

`err_ang_dirve:` margen para decidir si un robot ha llegado a su orientación final.

`max_byte:` máximo valor que puede tomar un dato que se va a enviar por el *transceiver*.

`radio_robot:` radio del robot (puede variar dependiendo de la altura de la cámara).

`dist_ref:` distancia que debe moverse la pelota a partir de un punto de referencia para cambiar automáticamente de estado de juego.

`tasa_camb:` tasa de cambio utilizada en el modelo de aceleración y desaceleración.

`dist_fren:` distancia de frenado y aceleración para la velocidad lineal.

`ang_fren:` ángulo de frenado para la velocidad de giro.

### 5.7. Simulador de movimientos

La interfaz de simulación permite modificar cualquiera de los parámetros utilizados dentro del simulador de movimientos:

`err_dist_sim:` rango utilizado para decidir si un robot ha llegado a su posición final.

`err_ang_sim:` rango utilizado para decidir si un robot ha llegado a su orientación final.La interfaz de simulación permite modificar cualquiera de los parámetros utilizados dentro del simulador de movimientos:

`err_dist_sim:` rango utilizado para decidir si un robot ha llegado a su posición final.

`err_ang_sim:` rango utilizado para decidir si un robot ha llegado a su orientación final.

`profundidad`: rango utilizado para decidir si se puede manipular la pelota con el *kicker* y el *dribbler*.

`angulo_frontal`: ángulo de apertura para definir si la pelota está frente al robot.

`aceleracion_pel:` desaceleración de la pelota por la fricción de la alfombra.

`tiempo:` periodo de simulación.

`elasticidad:` en el intervalo [0,1], define el porcentaje de velocidad que mantiene la pelota después de chocar con algún robot o porterías.

`vel_50%:` velocidad de simulación cuando la velocidad de control está al 50%.

`giro_50%:` velocidad de giro cuando la velocidad de control está al 50%.

### 5.8. Detección del movimiento de la pelota

La detección del movimiento de la pelota se implementó de forma distribuida entre el sistema de visión y el sistema de IA. El filtro de Kalman se implementó dentro del sistema de Visión. En cada ciclo de control el sistema de IA recibe la medición, corrección y predicción correspondientes y genera el vector de movimiento para ser utilizado en la estrategia de juego.

### 5.9. Estrategia de juego

Para la implementación del sistema experto en el sistema de IA se utilizó el lenguaje CLIPS. La definición de las reglas se realizó de forma independiente para cada uno de los roles por lo que se requieren cinco archivos por cada estrategia de equipo. En el sistema de IA se leen estos archivos y se insertan los hechos de acuerdo con la evaluación de las condiciones, utilizando la información del *referee box* y del sistema de visión. Se utilizó la librería clips.dll [18] que implementa el motor del sistema experto por medio de funciones que pueden invocarse desde la aplicación con el lenguaje C++. Esta librería utiliza la clase CCLIPSWrap.

A continuación se describe el proceso cómos e inserta el resultado de la evaluación del estado de juego en el sistema experto (tabla 5.2). En primer lugar se ejecuta el comando `clear()`, posteriormente se realiza la lectura de las reglas, se ejecuta el comando `reset()` y se inserta el hecho que tiene el valor del estado de juego actual `(estado_juego)`. El objeto `cond` regresa una cadena de caracteres con el nombre de dicho estado y toma como parámetro el objeto `game` que contiene el estado de juego actual. Finalmente, se ejecuta el

```
CLIPSEng.CLIPSClear();
IniciaReglas(archivo);
CLIPSEng.CLIPSReset();
strcpy(fact,"(estado_juego ");
strcat(fact,cond->EdoJuego(game));
strcat(fact,")");
str=fact;
if(CLIPSEng.CLIPSAssert(str)==FALSE)
      error=true;
CLIPSEng.CLIPSRun();
RealizaAccion(E,S,game,C,i);
```

***Tabla 5.2.*** *Inserción del hecho* `estado_juego` *utilizando la librería* Clips.dll.

comando `run()` y se ejecuta la acción resultante de la evaluación.

La función `IniciaReglas` se implementa tomando como parámetro el nombre del

```
void CClips::IniciaReglas(CString Script)
{
      int iErrCode;
      iErrCode = CLIPSEng.CLIPSLoad(Script);
   if(iErrCode != CCLIPSWrap::READ_OK)
   {
      printf("CLIPS failed while loading the script
%s\n", LPCSTR(Script));
      switch (iErrCode)
      {
         case CCLIPSWrap::READ_FAIL :
            AfxMessageBox("Read Failure\n");
            break;
         case CCLIPSWrap::PARSE_FAIL :
            AfxMessageBox("Parse Failure\n");
            break;
         case CCLIPSWrap::BAD_LOAD_NAME :
            AfxMessageBox("Bad Load Name\n");
            break;
         case CCLIPSWrap::READ_NOT_INIT :
            AfxMessageBox("CLIPS Not Initialized\n");
      }
      CLIPSEng.CLIPSExit(0);
   }
}
```

***Tabla 5.3.*** *Inicialización de un archivo de reglas.*

archivo (tabla 5.3).

La función `RealizaAccion` (Tabla 5.4) se encarga de obtener todos los hechos y buscar el último insertado que debe corresponder al nombre de la acción que se desea realizar. Finalmente, se ejecuta el comportamiento deseado por medio de la función `RealizaJugadaSistExp`, tomando como parámetros: la información actual de todos los robots `(CEscenario *E)`, la información sobre lo que se desea hacer `(CSuperEscenario *S)`, la información del *referee box* `(CGameControl *game)`, las dimensiones de la cancha `(CCancha *C)` y el número de robot que se está evaluando `(int i)`. El objeto `comp` tiene acceso a la implementación de todos los posibles comportamientos.

```
void CClips::RealizaAccion(CEscenario *E,
CSuperEscenario *S, CGameControl *game, CCancha *C,int
i)
{
      CStringArray facts;
      Char valor[100],fact[100];
      char *ap,*ap2;
      int n;
      bool resultado;

      strcpy(valor,"");
      CLIPSEng.GetAllFacts(facts);
      n=facts.GetSize();
      resultado=false;
      if(n>1)
      {
            strcpy(fact,"");
            strcpy(fact,facts[n-1]);
            ap=strstr(fact,"(accion ");
            if(ap!=NULL)
            {
                  ap2=strstr(ap," ");
                  ap2=ap2+1;
                  strcpy(valor,"");
                  strncat(valor,ap2,strlen(ap2)-1);
                  resultado=true;
            }
      }
      if(resultado)
            comp-
>RealizaJugadaSistExp(E,S,C,game,i,valor);
      else
            AfxMessageBox("No se encontró la acción a
realizarse");
}
```

***Tabla 5.4.*** *Obtención del último hecho para realizar la acción.*

### 5.10. Control de trayectorias

Los puntos utilizados para la navegación del robot se obtienen por medio de una librería que realiza la interpolación de los puntos de control. El algoritmo GET se implementó por medio de árboles binarios. Debido a que el tiempo de generación de los árboles de los cinco robots llega a ser mayor al tiempo que hay entre dos ciclos de control alentando la frecuencia a la que trabaja el sistema y afectando el control de los robots, que debe ser en tiempo real. Por esta razón se implementó la generación de un solo árbol en cada ciclo de control y se limitó el número de extensiones que puede tener un árbol para evitar que el árbol siga creciendo en casos en los que no existe solución.

# Capítulo 6

## PRUEBAS Y RESULTADOS

En este capítulo se describen las pruebas y los resultados obtenidos durante el desarrollo del sistema de IA. Este análisis se realizó de forma independiente para cada una de las funcionalidades del sistema, haciendo énfasis en aquéllas que requieren de la interacción con otros sistemas y del uso de algoritmos diseñados especialmente para su desarrollo. Estas funcionalidades son: control del movimiento, detección del movimiento de la pelota, estrategia de juego y control de trayectorias.

### 6.1. Visualización, ambiente de pruebas y simulador de movimientos.

La interfaz de control y el ontrolador externo se utilizaron como herramientas para realizar pruebas con los robots. La visualización en dos y tres dimensiones permitió comprender el funcionamiento del sistema para facilitar su desarrollo y el de sus módulos. Con el simulador de movimientos se probaron los algoritmos de control de movimientos, estrategia de juego, control de trayectorias y detección del movimiento de la pelota, sin necesidad de interactuar con los demás sistemas en un entorno controlado y sin ruido.

## 6.2. Control de movimientos

La metodología utilizada para controlar el movimiento de los robots es muy importante para lograr que realicen comportamientos autónomos y naveguen en el campo de juego. Para quela metodología funcione correctamente es necesario que el sistema de visión no presente fallas de calibración, que no exista interferencia en la comunicación entre la computadora y los robots, que los parámetros de los controladores del robot estén correctamente establecidos, que el funcionamiento de sus motores y electrónica en general no presente problemas y, finalmente, que la información enviada por el sistema de IA sea estable y factible de realizar por el robot.

Los parámetros utilizados en el sistema de IA se obtuvieron con la realización de pruebas con el sistema completo. Para evaluar este desempeño se tomó el comportamiento de patear la pelota en dirección a la portería contraria, el cual integra el uso de *splines*, requiere aproximación rápida y precisa hacia la pelota, así como el uso de los dispositivos del robot.

Los resultados obtenidos se aplican para cualquier comportamiento que pueda realizar un robot:

1. En caso de no utilizar el modelo de aceleración y desaceleración se introduce un error a la posición de partida del robot ocasionado por el derrape con el campo de juego, lo mismo ocurre para cambios drásticos de dirección.

2. La relación entre velocidad lineal y velocidad de giro es importante para lograr un movimiento controlado del robot. En caso de que la velocidad de giro sea muy alta se introducen errores en la trayectoria deseada.

3. Es necesario variar las velocidades lineal y de giro a lo largo del comportamiento con la finalidad de alcanzar una mayor precisión al final del recorrido.

4. Si la velocidad de giro es muy baja, existen casos en los que el robot no tiene la distancia suficiente para llegar a su orientación final por lo que llega a la pelota sin estar dirigido hacia ella y es necesario ajustar la trayectoria del *spline* para evitarlo.

5. Conforme aumenta la velocidad de movimiento del robot, el tiempo que transcurre desde que se detecta la posición del robot hasta que éste recibe la información correspondiente a dicha posición, introduce un retraso en el modelo de control.

6. El modelo de corrección PID global permite que la respuesta de los motores sea más eficiente y con menos errores a los vectores de movimientos enviados por la computadora que la obtenida cuando se aplica la corrección a cada uno de los motores.

7. Un problema que se presenta debido al peso de los robots, al modelo omnidireccional y al tipo de motores, es la dificultad para realizar movimientos con un grado de precisión

adecuado cuando los parámetros de los controladores PID del robot están configurados manualmente.

Una prueba que se utilizó para ajustar los movimientos de los robot fue la realización de círculos cuyo radio aumenta y disminuye en el tiempo variando la velocidad de los robots alrededor de un punto central del campo de juego, manteniendo fija la orientación del robot como se muestra en la secuencia de imágenes de la figura 6.1. De esta forma se prueban diferentes direcciones de movimiento para el modelo omnidireccional, tiempo de reacción al modificar la velocidad de movimiento y se pueden realizar ajustes en los parámetros PID del robot.

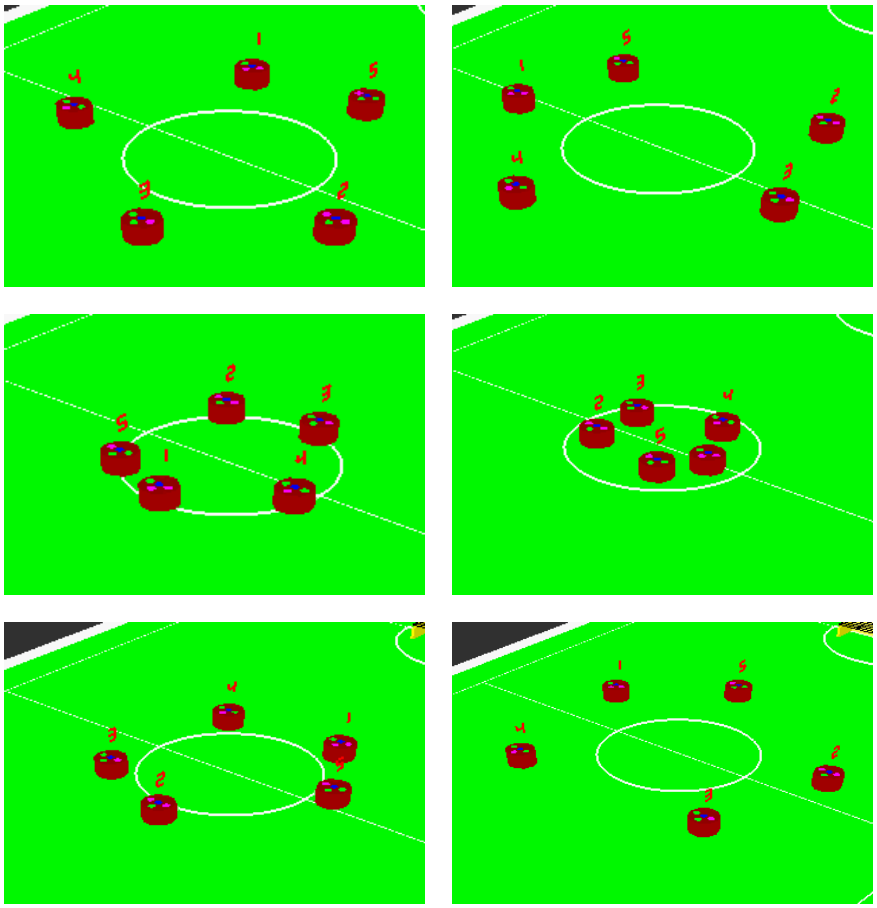


***Figura 6.1.*** *Movimientos en círculos para ajustar los parámetros utilizados en el control de movimientos.*

## 6.3. Detección del movimiento de la pelota

Para probar la eficiencia del filtro de Kalman en la obtención del vector de movimiento de la pelota y poder interceptarla, se utilizó la recepción de pase lateral descrita en la sección 4.5.7 entre los roles D1 y D2 sin obstáculos para medir el número de veces que el receptor logró alcanzar la pelota y meter gol.

Las veces que el receptor falló se debió principalmente a que el robot que le envió el pase, pateó la pelota muy desviada o porque el receptor no pudo colocarse para recibirla con la barra del *dribbler* para poder patearla nuevamente hacia la portería (tabla 6.1). Un problema es el tiempo que tarda el robot en recargar sus capacitores ya que se asume que la pelota viajará a cierta velocidad, pero si tiene tiempo de recargarse hay dos problemas: Si se espera a terminar la carga puede perder la oportunidad de enviar el pase con el impulso de su movimiento y si de cualquier forma lo da sin terminar la carga corre el peligro de que viaje más lento de lo debido.

| Total de pases | Número de veces que el receptor alcanzó la pelota | Número de veces que el receptor anotó gol |
|---|---|---|
| 10 | 9 | 7 |

***Tabla 6.1.*** *Resultados de las pruebas de intercepción de la pelota.*

Se determinó que lo ideal para definir el vector de movimiento es una predicción de tres estados. En caso de tomar un número menor, la predicción no se separa tanto de la medición por lo que el vector de movimiento tiende a variar en un mayor rango. Esto ocurre porque la variación entre dos mediciones consecutivas (error de medición) es mayor que la distancia utilizada para generar el vector. Si se toman más estados, la predicción se separa demasiado y los errores del modelo tienen mayor repercusión.

La principal falla proveniente del diseño del filtro de Kalman es que hace falta integrar al modelo las colisiones con los robots y las porterías que ocasionan cambios repentinos en el movimiento de la pelota, aunque esto no es determinante para el buen funcionamiento del filtro. Mientras la dirección en que el robot recibe la pelota con respecto a su eje central sea mayor las posibilidades de que reciba correctamente la pelota disminuyen. Es por esta razón que los comportamientos de envío y recepción explicados en la sección 4.5.9 se desencadenan en condiciones que facilitan el ángulo de recepción.

En las figuras 6.2 a 6.6, se muestra un ejemplo del portero interceptando la pelota cuando el equipo contrario realiza un pase de un lado al otro del campo de juego [35].


***Figura 6.2.*** *Intercepción de la pelota por el portero (Parte 1).*

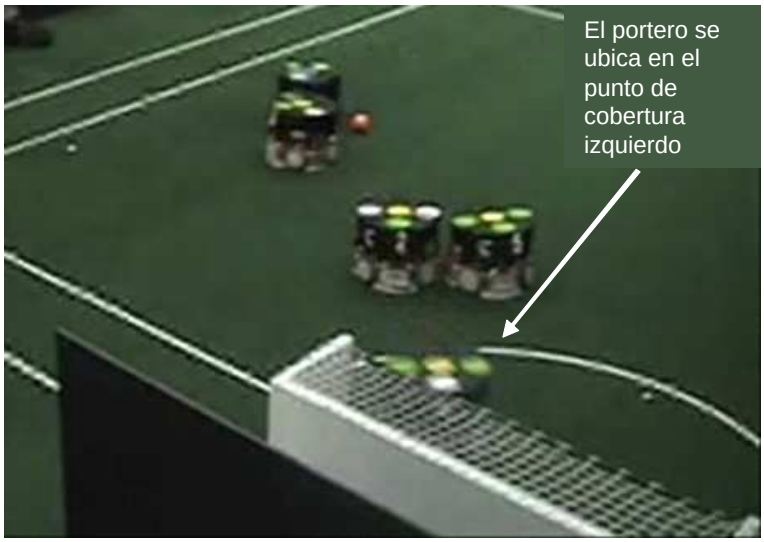



***Figura 6.3.*** *Intercepción de la pelota por el portero (Parte 2).*

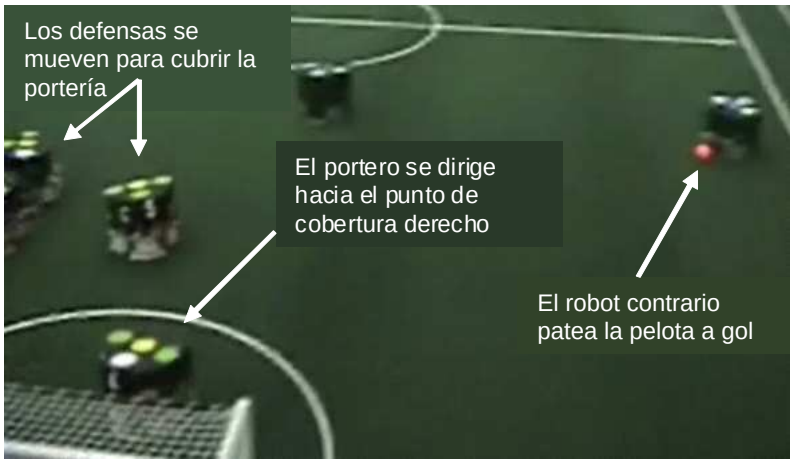



***Figura 6.4.*** *Intercepción de la pelota por el portero (Parte 3).*


***Figura 6.5.*** *Intercepción de la pelota por el portero (Parte 4).*


***Figura 6.6.*** *Intercepción de la pelota por el portero (Parte 5).*

## 6.4. Estrategia de juego

Para analizar los resultados de la estrategia de juego basada en un sistema experto se tomó como punto de comparación la implementación de árboles binarios para decidir las acciones que realiza cada robot.

El uso de árboles binarios ha sido una primera aproximación para resolver el problema de la estrategia de juego en el equipo EK. Anteriormente, la estructura de las preguntas binarias se realizaba directamente en código por medio de una serie de preguntas IF-THEN. Para estrategias en donde el espacio de búsqueda es reducido, esta solución no presenta ningún problema. Cuando el problema de la búsqueda cumple con las características de tener que representar el conocimiento en un dominio específico, se presentan complicaciones en el proceso de búsqueda de la solución.

Por ejemplo, si el rol defensa puede participar en la barrera de la portería bajo ciertas condiciones en el estado START, pero también lo puede hacer durante otros estados de juego, las preguntas realizadas para escoger la forma en que participa en la barrera se repiten en varias secciones del árbol. Otro problema es que si se modifica esta condición,

se tienen que buscar en el resto del árbol todos los casos en los que el robot puede participar en la barrera y modificarlos de la misma forma. Si se utiliza una misma condición para decidir entre dos posibles caminos para dos roles distintos, ésta debe ser evaluada de la misma forma en cada parte del árbol para evitar inconsistencias.

Finalmente, al realizar la deducción desde la raíz hasta las hojas del árbol se puede presentar el caso en el que al evaluar un nodo, se desee regresar a otro nodo del árbol. Esto implica tener que repetir toda esa sección debajo del nodo evaluado.

La principal aportación que ofrece la implementación de un sistema experto para definir la estrategia de juego de los cinco robots es que se pueden formular formas de juego en equipo más complejas y de una forma más intuitiva que por medio de árboles de búsqueda. Conforme aumenta la complejidad de la estrategia, se agregan reglas que pueden considerar diferentes etapas en el proceso de la toma de decisión. Un esquema basado en máquinas de estados o árboles de búsqueda, resultaría en la repetición de los casos y las condiciones que se deben tomar en cuenta.

En conclusión, pequeñas modificaciones implican grandes cambios en la estructura del árbol que pueden ocasionar inconsistencias en las evaluaciones realizadas para determinar la acción de los robots y dificultan el modelado del conocimiento para este problema específico. Durante un partido se debe contar con una estrategia clara para el usuario, ya que cualquier cambio permitido durante un tiempo fuera en el partido, debe ser preciso y sin errores que puedan poner en peligro el resultado del mismo.

Las figuras 6.7 a 6.9 muestran un ejemplo de la transición entre el estado de juego STOP y un tiro directo a nuestro favor [35].

Las figuras 6.10 a 6.13 muestran una secuencia de figuras que describe la realización de un pase entre dos robots integrando la estrategia definida en el sistema experto y la intercepción de la pelota por medio del filtro de Kalman.


***Figura 6.7.*** *Cambio de estados de juego (Parte 1).*

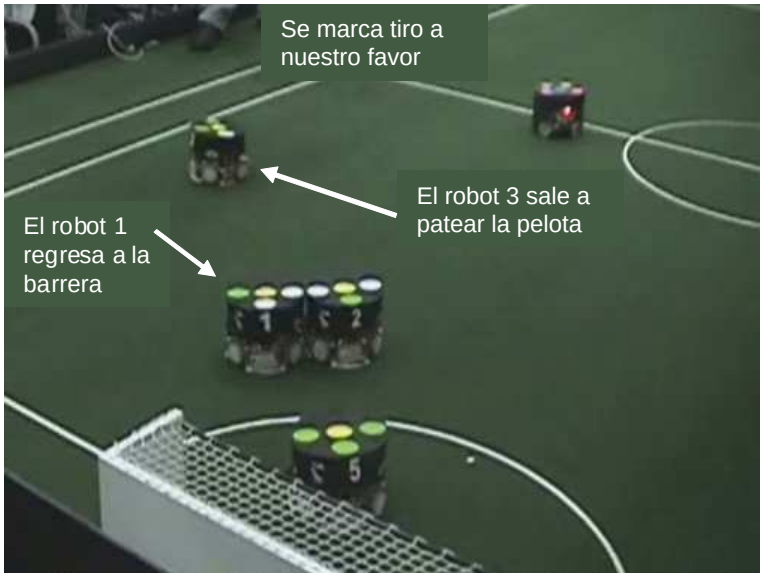



***Figura 6.8.*** *Cambio de estados de juego (Parte 2).*


***Figura 6.9.*** *Cambio de estados de juego (Parte 3).*


***Figura 6.10.*** *Realización de pase (Parte 1).*

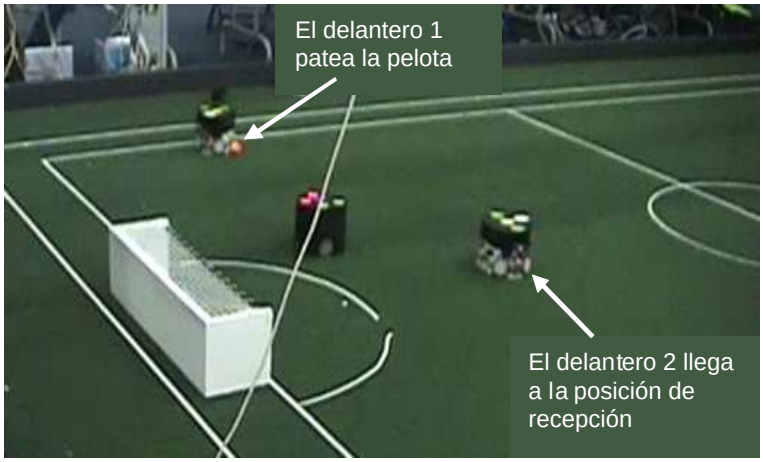



***Figura 6.11.*** *Realización de pase (Parte 2).*


***Figura 6.12.*** *Realización de pase (Parte 3).*


***Figura 6.13.*** *Realización de pase (Parte 4).*

## 6.5. Control de trayectorias

Para el diseño de los algoritmos utilizados en el control de trayectorias se siguió el siguiente proceso:

1. Planeación e implementación del algoritmo.
2. Prueba del los algoritmos con ayuda del simulador de movimientos y el ambiente gráfico.
3. Prueba de los algoritmos por medio del ambiente de pruebas interactuando con el sistema de visión y los robots.

A continuación se describen las pruebas y resultados obtenidos para las trayectorias basadas en splines *y la planeación de rutas por medio de árboles GET.*

### 6.5.1. Trayectorias basadas en splines

El uso de *splines* para seguir puntos de control de forma suave permitió resolver correctamente el problema de aproximación a la pelota, incluso cuando se encuentra en movimiento. En este caso, los puntos de control se modifican en función de la posición de la pelota y la trayectoria del *spline* se genera partiendo del robot hasta su coordenada final.

En algunos casos se observó que el robot no alcanza a girar para llegar orientado correctamente a su ángulo final, debido a que los valores de las velocidades de traslación y de giro se ajustan manualmente. Para solucionar este problema es necesario tomar en cuenta la respuesta real del robot para una cierta velocidad definida. De esta manera es posible modelar la trayectoria y definir las velocidades de traslación y de giro, conociendo con certeza que el robot tendrá el tiempo necesario para realizar la acción correctamente.

Un factor que afecta el desempeño de la navegación del robot es la resolución utilizada para evaluar los puntos de la trayectoria, ya que si la distancia entre cada punto es muy grande se pierde la suavidad de la interpolación, pero si es muy chica requiere un mayor procesamiento, sobre todo si la longitud de la trayectoria es muy larga.

Para definir la mejor forma de navegación del robot, se realizaron pruebas directamente en el simulador y, una vez que el comportamiento ideal del robot fue el correcto, se procedió a probarlo con el robot físicamente. Para probar el funcionamiento del algoritmo se utilizaron *splines* con diferentes variaciones en su implementación:

1. En la **primera prueba,** el *spline* se recalcula en cada ciclo de control considerando al propio robot como un punto de control y se calcula el vector de movimiento entre el robot y el siguiente punto de la trayectoria del *spline*. El principal problema es determinar el orden de los puntos de control al generar los puntos de la trayectoria. Suponiendo que existen cuatro puntos de control: una coordenada inicial, la coordenada robot, la coordenada detrás de la pelota (en la dirección de tiro deseada) y la pelota misma. Es difícil determinar en qué momento el robot deja de estar entre la coordenada inicial y la coordenada detrás de la pelota, para colocarse entre la coordenada detrás de la pelota y la pelota, considerando que la generación de la trayectoria varía dependiendo de la posición del robot. Otro problema que se presenta es que, por restricciones físicas, el movimiento del robot no sigue la trayectoria ideal y este error distorsiona la generación de la propia trayectoria.

2. En la **segunda prueba**, en lugar de considerar al propio robot como un punto de control como en el caso anterior, se generó la trayectoria del *spline* a partir de tres puntos de control: la coordenada inicial, la coordenada detrás de la pelota y la pelota misma. En lugar de generar la trayectoria en cada ciclo de control, ésta se genera únicamente cuando la coordenada de la pelota varía de un ciclo de control a otro en una distancia mayor a una constante predefinida. El resultado de lo anterior es que la trayectoria se mantiene fija a menos que la pelota esté en movimiento. Para saber qué vector de movimiento debe seguir el robot, se calculó el punto más cercano al robot de la trayectoria del *spline* y el siguiente punto de la trayectoria. En este caso, el robot no distorsiona la trayectoria a seguir, pero permanece el problema de que por restricciones físicas no puede seguir la trayectoria ideal

y, por lo tanto, no llega con la precisión requerida a la pelota, aunque en ciertos casos sí logra aproximarse correctamente.

3. La **tercera prueba** fue la mejor aproximación para lograr que el robot navegara siguiendo la trayectoria del *spline* y es la utilizada en la sección 4.6. En este caso, la trayectoria del *spline* se calcula en cada ciclo de control. Se utilizaron tres puntos de control: la coordenada del robot, la coordenada detrás de la pelota y la pelota misma. El vector de movimiento se calcula entre el primer punto (que siempre es la coordenada del robot) y el segundo punto de la trayectoria. Para determinar el momento en el que el robot sobrepasa un punto de control, se calcula la diferencia entre dos vectores de movimiento consecutivos y si esta diferencia es demasiado grande, significa que se rebasó el siguiente punto de control por lo que hay que eliminarlo. Un caso particular es cuando el robot se encuentra muy cerca de la pelota y es preferible que se mueva directamente hacia ella en lugar de seguir la trayectoria del *spline*. Las figuras 6.14-A y 6.14- B muestran una secuencia de imágenes de la aproximación del robot a la pelota para tirar a gol generando la trayectoria por medio de *splines*.

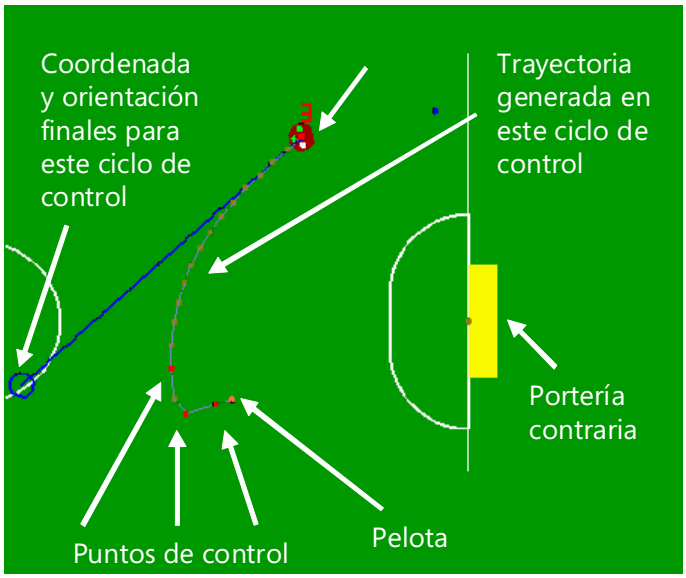



***Figura 6.14-A.*** *Aproximación a la pelota para tirar a gol por medio de splines.*

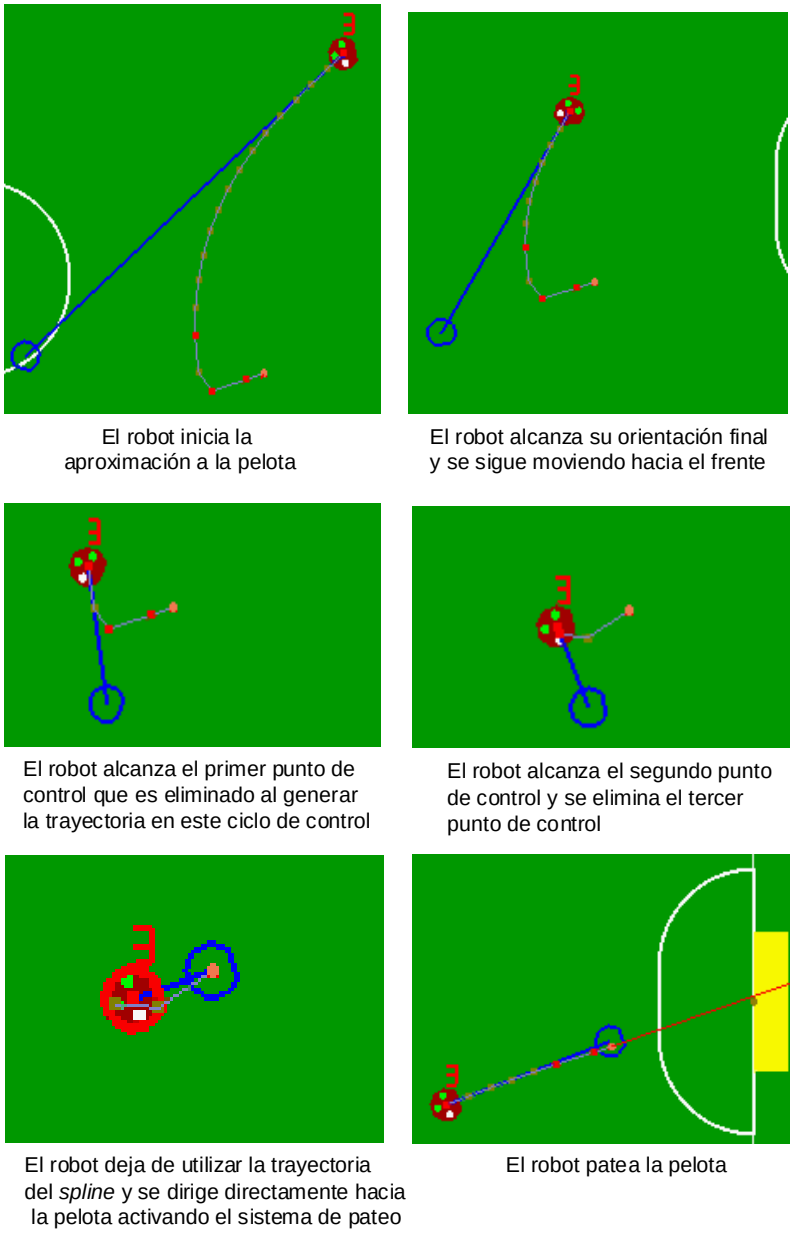



***Figura 6.14-B.*** *Secuencia de navegación por medio de splines.*

### 6.5.2. Planeación de rutas por medio de árboles GET

Para evaluar el desempeño de los árboles de exploración geométrica, se realizó inicialmente una implementación de los árboles de exploración aleatoria. Los parámetros para evaluar la eficiencia de cada algoritmo radican en el tiempo de generación del árbol, la facilidad de navegación, así como la respuesta del algoritmo ante circunstancias complejas. También se realizó una comparación con el algoritmo de campos potenciales utilizado para la evasión de obstáculos [13].

La figura 6.15 muestra el ejemplo del árbol RRT aplicado al problema de los robots SSL, en donde se modelaron los obstáculos con un radio igual al doble del radio de un robot, el navegador como punto A y la meta como punto B.

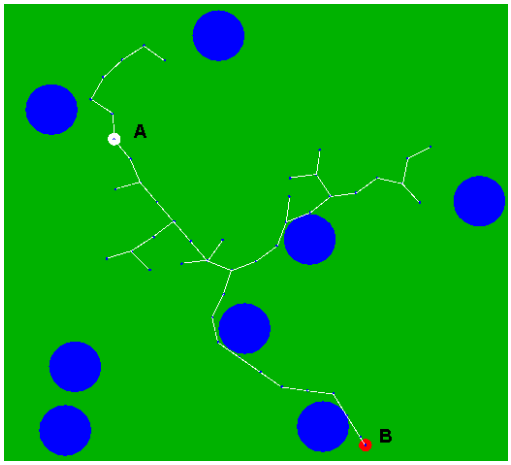



***Figura 6.15.*** *Generación de un RRT en un ambiente simulado.*

La figura 6.16 ilustra la trayectoria seguida por el robot navegador desde su punto de inicio hasta la meta. Como se observa, la forma de los árboles es distinta en cada etapa debido a la aleatoriedad del algoritmo.

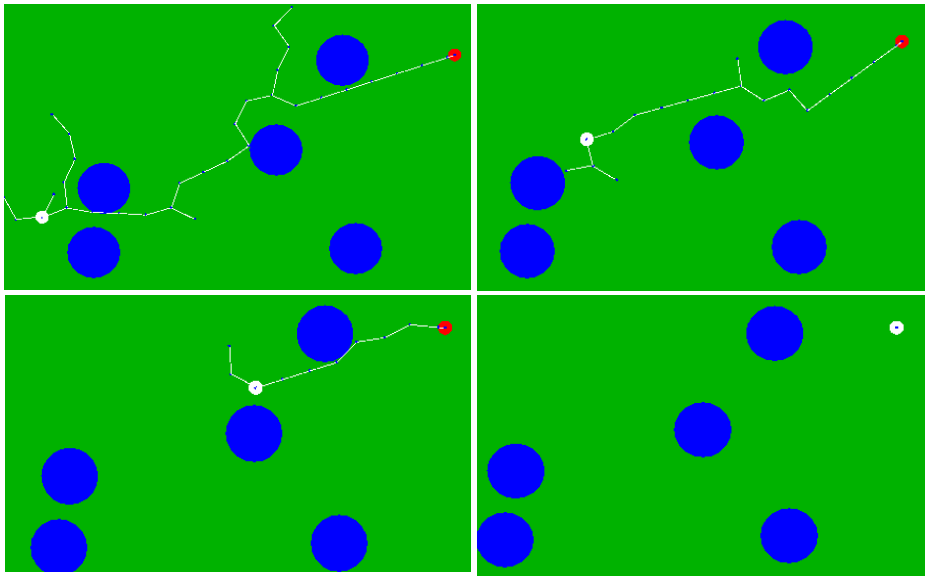


***Figura 6.16.*** *Seguimiento de una trayectoria trazada por un árbol RRT.*

A continuación se discuten los resultados obtenidos a partir de las pruebas realizadas con los diferentes algoritmos. Para este análisis, se entiende por configuración sencilla aquella en la que el robot puede llegar a su meta sin que los obstáculos representen una dificultad real; por configuración compleja aquella en la que existen varios obstáculos cercanos entre sí que se tienen que evadir para llegar correctamente a la meta y por configuración sin solución, aquella en la que no existe una ruta libre de obstáculos entre el robot y la meta.

La inestabilidad es la dificultad que experimenta un robot para seguir congruentemente una ruta hacia su meta y caer en un estado de duda entre varias posibles soluciones.

**Resultados obtenidos para el algoritmo RRT:**

1. El aumento en el tamaño de las extensiones resultó en la disminución del tiempo de generación del árbol, aunque también aumentó la falta de suavidad en la trayectoria durante la navegación del árbol.

2. Se observó que conviene que el valor de probabilidad $p$ para extender el árbol hacia la meta conviene que sea muy alto para configuraciones sencillas y muy bajo para configuraciones complejas. Existen configuraciones muy complejas o sin solución en las

que el tiempo de generación del árbol aumenta exponencialmente por lo que es necesario limitar el número de extensiones para evitar que el algoritmo retrase el proceso de control de los robots.

3. Existen casos de inestabilidad difíciles de compensar por la aleatoriedad del algoritmo.
4. No se utilizó el algoritmo en competencias oficiales.

**Resultados obtenidos para el algoritmo de campos potenciales:**

1. El tiempo de generación del algoritmo depende del número de máscaras que existen en el campo y se mantiene constante independientemente de la configuración del campo de juego.
2. Existen casos de inestabilidad en el tiempo para configuraciones complejas o sin solución que se resuelven correctamente al planificar la ruta con el algoritmo GET.
3. Para lograr una mayor suavidad en el movimiento del robot es necesario aumentar la resolución de las máscaras, resultando en un aumento del tiempo de procesamiento, aunque se mantuvo dentro de las restricciones del tiempo real.
4. No es posible conocer con anterioridad la ruta que debe tomar el robot, lo que dificulta verificar el funcionamiento eficiente del algoritmo.
5. Se probó el algoritmo en competencias oficiales del 2003 al 2005.

**Resultados obtenidos para el algoritmo GET:**

1. Se determinó que tanto para configuraciones sencillas como complejas, el tiempo de generación del árbol es menor al RRT.
2. Es posible una mayor reducción en el tamaño de las extensiones que con RRT porque el tiempo de procesamiento es menor y se puede lograr una mayor suavidad en el movimiento del robot.
3. Pueden darse casos en los que el árbol requiere crecer demasiado por lo que es conveniente limitar el número de extensiones para lograr que el algoritmo funcione en tiempo real.
4. Es posible solucionar los casos de inestabilidad del algoritmo, gracias al registro que se lleva del robot cuando está evadiendo un obstáculo y del sentido en el que lo hace.
5. Se probó el algoritmo en las competencias oficiales del 2006 al 2008.

La realización de pruebas de este algoritmo se realizó en varias etapas. La primera consistió en que un robot siguiera la pelota la cual se puso en movimiento entre una serie de obstáculos como lo muestran las figuras 6.17 a 6.21.

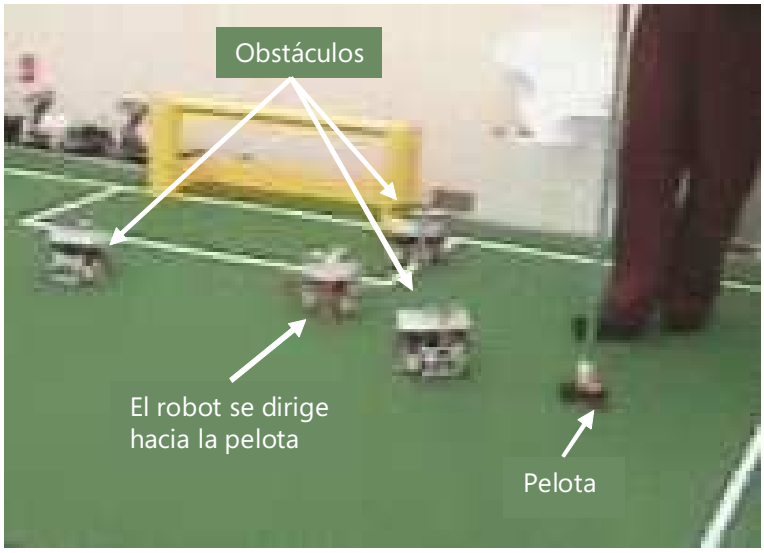



***Figura 6.17.*** *Ajustes de los parámetros de aproximación del algoritmo GET (Parte 1).*

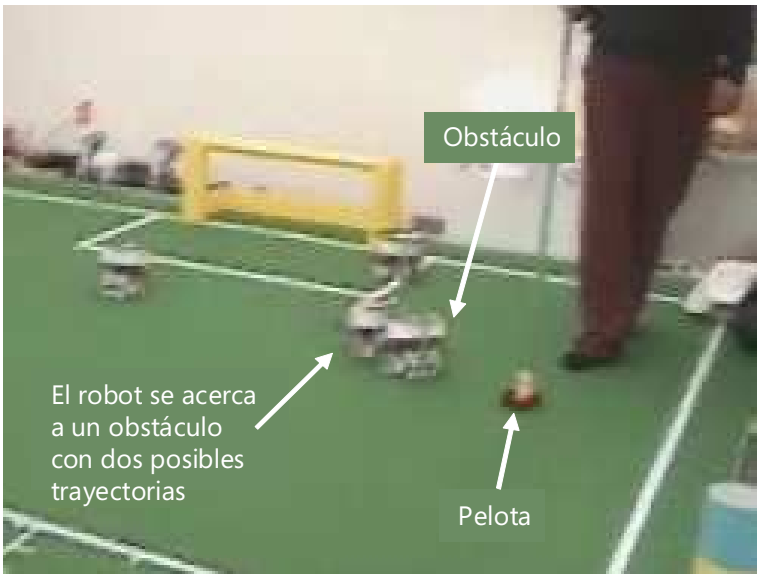



***Figura 6.18.*** *Ajustes de los parámetros de aproximación del algoritmo GET (Parte 2).*


***Figura 6.19.*** *Ajustes de los parámetros de aproximación del algoritmo GET (Parte 3).*

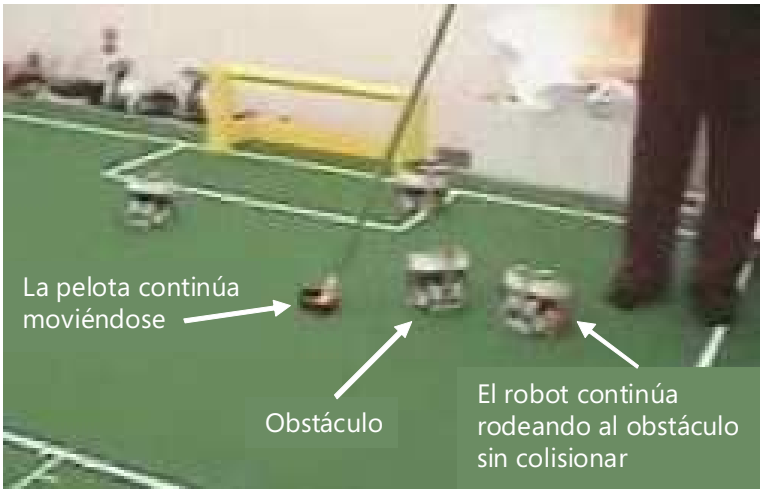



***Figura 6.20.*** *Ajustes de los parámetros de aproximación del algoritmo GET (Parte 4).*


***Figura 6.21.*** *Ajustes de los parámetros de aproximación del algoritmo GET (Parte 5).*

Una vez ajustados los radios de aproximación para la velocidad de movimiento del robot, se colocaron configuraciones como las descritas previamente para finalmente probarse en un partido con otros nueve robots en juego en configuraciones similares a las explicadas en la sección 4.7.

Las figuras 6.22-A y 6.22-B muestran una secuencia de imágenes de cómo realizan la evasión de obstáculos cuatro robots cruzando el campo de juego simultáneamente por medio de árboles GET.

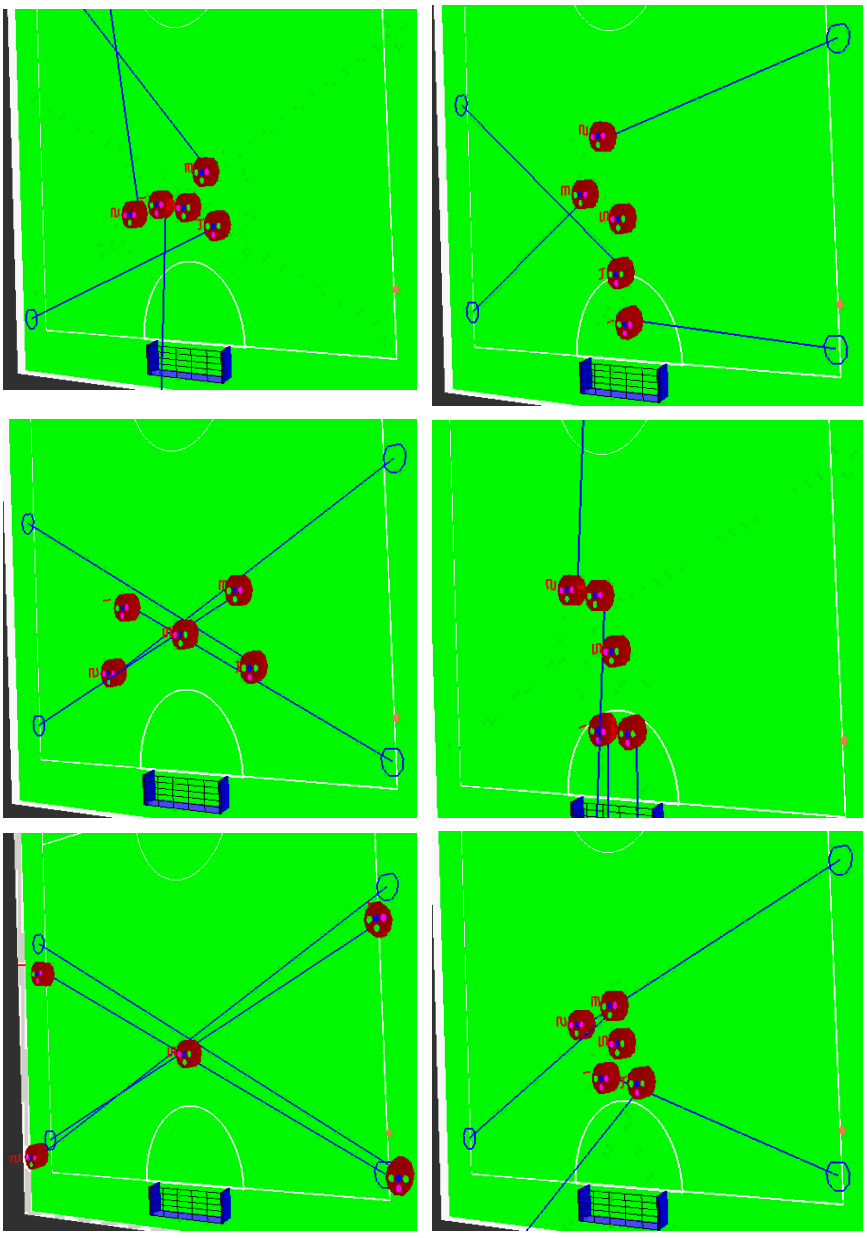


*Figura 6.22-A. Algoritmo GET para cinco robots simultáneamente (primera parte).*

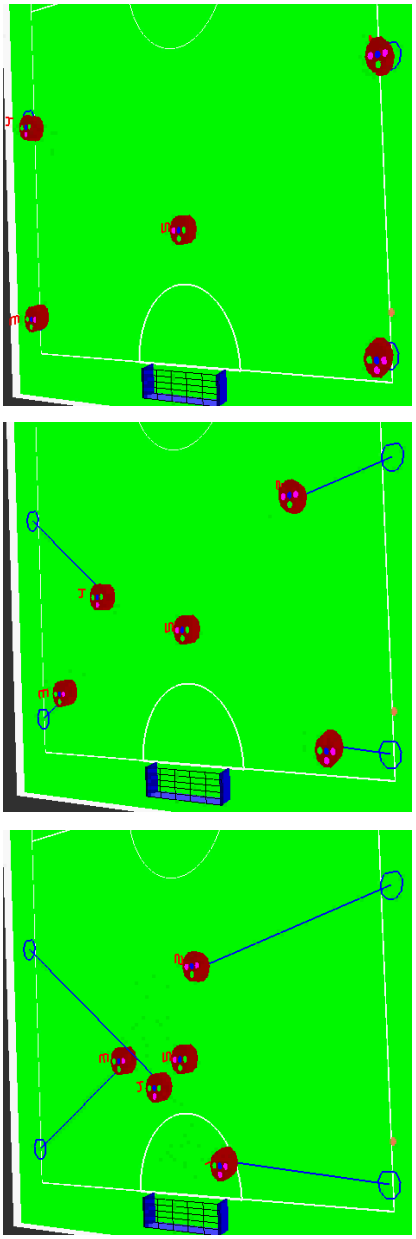


*Figura 6.22-B. Algoritmo GET para cinco robots simultáneamente (segunda parte).*

# Capítulo 7

## Conclusiones

El sistema de IA ha contribuido en el éxito logrado por el equipo *Eagle Knights SSL* del Laboratorio de Robótica del ITAM, obteniendo el primer lugar en el *1st Latin American Open*, el séptimo lugar en el mundial de Atlanta 2007 y ha sido utilizado en las competencias mundiales del *RoboCup* del 2006 al 2008.

Los avances en el desarrollo de este sistema junto con los del sistema de visión, han permitido tener una plataforma estable y confiable para el desarrollo mecánico y electrónico de los robots y para la participación en futuras competencias mundiales de *RoboCup*. El sistema sirve también como una plataforma de apoyo y referencia para el trabajo realizado en la materia de Robótica del Departamento de Sistemas Digitales y para otros proyectos del Laboratorio de Robótica.

El presente trabajo de tesis cumplió con sus objetivos resolviendo todos los requerimientos planteados; manteniendo una arquitectura flexible que permite fácilmente adaptarse a cambios en las reglas, las dimensiones del campo de juego y la altura de las cámaras; contando con un ambiente de pruebas robusto y una estrategia de juego basada en un sistema experto y mejorando radicalmente el control de movimiento implementado en forma distribuida en el robot y en el sistema de IA.

La implementación de este sistema se llevó a cabo tomando en cuenta la optimización en el tiempo de procesamiento de los algoritmos y realizando todas las pruebas necesarias para validar el correcto funcionamiento de los algoritmos explicados en este trabajo.

## 7.1. Visualización y ambiente de pruebas

El proceso de desarrollo del sistema de IA fue especialmente difícil por la gran interacción e interdependencia con el sistema de visión y los robots. El ambiente gráfico fue de gran utilidad para probar y mejorar las funciones realizadas en el sistema de visión. El ambiente de pruebas le permite al usuario validar el correcto funcionamiento del *hardware* y *software* del robot. Es difícil probar y mejorar la arquitectura de los robots sin estos módulos.

## 7.2. Control de movimientos

El uso de los controladores PID aplicados directamente a los vectores de movimiento del robot, en lugar de hacerlo de forma independiente en cada motor, presentó mejoras en el control del robot y facilitó la programación de los diferentes comportamientos en el sistema de IA. La forma de obtener los parámetros de los controladores PID puede ser optimizada por medio de un algoritmo de aprendizaje por refuerzo para reducir el rango de error al seguir diferentes trayectorias [16]. Lo anterior consiste en utilizar el sistema de visión para medir el error en los movimientos del robot y variar los parámetros de los controladores para obtener mejores resultados.

Una mejora radical para lograr mayor precisión en los movimientos del robot a muy bajas velocidades relacionada con el *hardware* sería cambiar el tipo de motores utilizados.

Conforme aumenta la velocidad de control de los robots también se incrementan los problemas para el control autónomo de los mismos, lo que mejoró al incorporar el modelo de aceleración y desaceleración; sin embargo, se introduce el error de la latencia del sistema, ya que mientras más rápido se mueve el robot, la distancia que logra avanzar en el tiempo que pasa entre la captura de la imagen y el momento en el que el robot reacciona ante la información proveniente de este dicha imaten, ocasionando que el robot tenga un retraso en la ejecución de sus acciones.

Otro punto importante para mejorar el control del robot es la realización de movimientos a velocidades bajas con un alto grado de precisión. El principal problema actualmente es que los motores del robot están directamente sujetados a las ruedas omnidireccionales. El uso de engranes entre el motor y la rueda permite distribuir la misma cantidad de potencia utilizada por el motor para realizar un movimiento de menor distancia en la rueda aumentando la precisión y disminuyendo la velocidad máxima de movimiento, la cual

actualmente se encuentra muy por encima de las capacidades de control autónomo impuestas por las características de la arquitectura completa.

### 7.3. Simulador de movimientos

El uso de un simulador permitió visualizar las acciones y la forma en que los robots las realizan. Se diseñó de forma flexible para poder mover a los robots a diferentes velocidades y variando los parámetros utilizados para acercarse lo más posible a la realidad. Sin embargo, el simulador se desarrolló basándose en las leyes teóricas del movimiento omnidireccional por lo que una vez validadas las acciones en el simulador para resolver los problemas más comunes es necesario ajustarlas directamente con los robots y con el sistema de visión, especialmente porque el robot reacciona de una forma distinta a la teóricamente supuesta. Esto es un problema porque no hay disposición de los robots todo el tiempo y para poder programar una buena estrategia de juego se requiere tener a más de un robot funcional a lo largo de la programación de la estrategia. Esto se puede resolver integrando un simulador que aprenda el movimiento real del robot como se propone en la sección 7.2.

### 7.4. Detección del movimiento de la pelota

EL filtro de Kalman utilizado para el movimiento de la pelota permite obtener predicciones basadas en un proceso recursivo en el que se minimiza el error de dicho modelo conforme se obtienen mediciones dinámicamente. Este modelo puede adaptarse para ser utilizado no sólo en la pelota, sino extenderse también al control de nuestros robots, utilizando los vectores de movimiento enviados al robot en el parámetro de control del filtro de Kalman. Este modelo podría utilizarse en el control de movimientos con la finalidad de aprovechar la información proveniente del sistema de visión para definir el vector de movimiento del robot.

### 7.5. Estrategia de juego

La integración de un sistema experto para la estrategia de juego permite la programación y prueba de nuevas estrategias para los diferentes tipos de rivales y seguir adecuando el

funcionamiento del sistema a los avances que el *hardware* del robot vaya permitiendo. Es una forma más conveniente para coordinar a los robots por encima de los árboles de búsqueda y máquinas de estados. Para mejorar la forma de juego es necesario incrementar la potencia de pateo de los robots para poder meter gol con mayor facilidad y realizar pases, así como poder moverlos a mayor velocidad. Se observó también que las limitaciones físicas en el sistema de pateo de los robots no facilitan la emisión y recepción de pases. En las competencias se determinó que es conveniente contar con al menos tres estrategias de juego, dependiendo del nivel del rival: avanzado, intermedio y básico; por ejemplo, es mejor tirar constantemente a gol que utilizar una estrategia de pases y marcación cuando el equipo contrario tiene muchos problemas con sus robots, pero esta estrategia sería deficiente para equipos muy competitivos.

## 7.6. Control de trayectorias

La generación de trayectorias por medio de *splines* permitió mejorar la ejecución de los comportamientos de los robots definidos en la estrategia de juego. En cambio la planeación de rutas por medio de árboles GET es un algoritmo que puede usarse independientemente del comportamiento a ejecutarse y la principal finalidad que ofrece es la evasión de obstáculos en un ambiente dinámico, con ruido e impredecible.

### 7.6.1. Trayectorias basadas en splines

El uso de *splines* permitió realizar un mayor número de comportamientos más interesantes y de mejor forma como los *zig-zags*, las trayectoria en forma de ocho y la aproximación a la pelota (en dirección a la portería contraria o para dar un pase). Esta última aprozimación había sido siempre un problema, ya que requiere movimientos a velocidades altas para llegar rápidamente a la pelota y bajas para patear en la dirección correcta, integrando movimientos precisos y rápidos junto con el uso de los dispositivos del robot.

Los comportamientos como robarle la pelota a un robot contrario o ir por la pelota y girar con ella sujetándola con el *dribbler* son acciones que aún son difíciles de ajustar a las restricciones físicas del robot, especialmente por el grado de precisión requerido en la mecánica utilizada para sujetar las ruedas y el tipo de motores utilizados.

### 7.6.2. Planeación de rutas por medio de árboles de exploración geométrica

El algoritmo GET empleado para la planeación de rutas es una solución innovadora, tomando en cuenta las principales problemáticas presentes en la navegación de los robots e incorporando un trabajo de investigación en las diferentes técnicas utilizadas para la evasión de obstáculos como los árboles RRT y los campos potenciales. De esta manera, ha sido posible proponer una solución adecuada para este problema en particular, permitiendo a los robots resolver configuraciones complejas durante los partidos y mejorar su dinámica de juego en equipo.

## 7.7. Líneas futuras

Para facilitar la programación de las acciones de los robots se pueden realizar diferentes mejoras en el sistema de IA. Una primera línea de trabajo sería el desarrollo de un simulador que mediante el aprendizaje de la respuesta del robot por medio del sistema de visión, integre las restricciones impuestas por las características físicas de los robots como la fricción, el peso y el tiempo de reacción, permitiendo crear un modelo apegado a la realidad. Este simulador podría ser utilizado para desarrollar la estrategia de juego y programar los comportamientos sin necesidad de hacerlo físicamente con los robots, cuya disponibilidad depende del tiempo de las baterías y del mantenimiento de sus sistemas mecánico y electrónico. Otra ventaja que ofrece este tipo de simulador es que permite predecir con mayor precisión el movimiento real de los robots, lo que podría ser utilizado para contrarrestar la latencia en el tiempo de procesamiento del sistema completo y así controlar a los robots a una mayor velocidad. Finalmente, el simulador permitiría automatizar las velocidades de los vectores de movimientos lineal y de giro, que por el momento son definidas manualmente según el comportamiento que se desea que el robot realice, tomando como referencia la respuesta real que ofrece este modelo.

Una propuesta para la realización de este simulador sería que a partir de un entrenamiento que tomara como datos de entrada las trayectorias seguidas por el robot para diferentes vectores de movimiento con base en las mediciones del sistema de visión mediante el uso de una red neuronal [32], se recreara la respuesta del robot para distintos vectores y velocidades de movimiento.

Una segunda línea de trabajo sería la modificación del sistema de comunicación inalámbrica entre la computadora y los robots para disminuir los problemas de interferencia.

Esto podría lograrse integrando transmisores con más de un canal de frecuencia, lo que requiere modificaciones en el *hardware* de los robots y en la interfaz de comunicación con la computadora que transmite la información.

Una tercera línea de trabajo sería mejorar la arquitectura mecánica de los robots, diseñando un modelo que incluya el peso, el tipo de motores y el uso de engranes para lograr la realización de movimientos más lentos y finos que requieren mayor torque en las ruedas. Esto mejoraría radicalmente la forma en que se ejecutan los comportamientos y la interacción de los robots en equipo.

Una cuarta línea de trabajo para mejorar el control omnidireccional de los robots sería integrar la información proveniente de un acelerómetro y un giroscopio que permiteran tener una medición más certera de la lectura del movimiento del robot que la obtenida utilizando únicamente la información proveniente de los *encoders* de los motores.

Finalmente, es importante decir que cualquier avance que se realice en este sistema debe considerar si la problemática surge en el sistema de IA, en el sistema de visión o en los robots. El aumento en las capacidades y estabilidad de los robots es un punto clave para mejorar la calidad del juego, tomando en consideración el funcionamiento actual del sistema de IA y del sistema de visión.

# Anexo

### Instalación del Sistema de IA

El sistema de inteligencia artificial “EKIntel” está programado sobre MFC de Microsoft Visual C++ 6.0. El programa puede estar ubicado en cualquier carpeta del sistema y utiliza archivos de texto de la carpeta “C:\Robotica”.

**a) Configuración del puerto paralelo para el uso del *transceiver*.**

1. Asegurarse de que el archivo io.dll se encuentre en la carpeta *Debug* del proyecto.
2. Entrar al menú del BIOS y configurar el puerto paralelo de la siguiente forma:

Modo: ECP
Base I/O Address: 378
Intemp: IRQ 7
DMA: DMA0
(Las direcciones de memoria de cada línea del puerto se establecen a partir de la 0x379.)

**b) Configuración de gráficas con OpenGL**

1. Descargar los siguientes archivos: glut.h, glut.lib, glut.dll y guardarlos en las siguientes carpetas:

| | |
|---|---|
| glut.h | C:\Program Files\Microsoft Visual Studio\VC98\Include\GL |
| glut.lib | C:\Program Files\Microsoft Visual Studio\VC98\Lib |
| glut.dll | *C:\WINDOWS\system32* |

**2. Dentro del proyecto en Visual C++ ir a:**

Project/Settings/Link

Añadir los siguientes archivos en la parte de Object/library modules

opengl32.lib glu32.lib glut32.lib

(el último no se utiliza en el proyecto, pero es conveniente para otras aplicaciones de gráficas en opengl)

**c) Configuración del uso del *joystick* con DirectX**

1. Descargar el paquete completo de DirectX 9.0 Sdk e instalarlo (no sólo un update, sino todo el paquete)

2. Dentro del proyecto en Visual C++ ir a:

Project/Settings/Link

Añadir el siguiente archivo en la parte de Object/library modules

dinput8.lib

Finalmente es necesario añadir las siguientes rutas para los directorios que se usarán en el siguiente orden:

Carpeta de directX

C:\DXSDK\INCLUDE

Arhivos generales de Visual C++ (por default)

C:\Archivos de programa\Microsoft Visual Studio\VC98\INCLUDE

C:\Archivos de programa\Microsoft Visual Studio\VC98\MFC\INCLUDE

C:\Archivos de programa\Microsoft Visual Studio\VC98\ATL\INCLUDE

Carpeta de opengl

C:\ARCHIVOS DE PROGRAMA\MICROSOFT VISUAL STUDIO\VC98\INCLUDE\GL

**d) Uso de otras fuentes en la interfaz gráfica del programa.**

1. Descargar el tipo de fuente deseado en la carpeta:

C:\WINDOWS\Fonts

**e) Configuración de la librería clips**

1. Asegurarse de que el archivo clips.dll se encuentre en la carpeta del proyecto.
2. Dentro del proyecto en Visual C++ ir a:
Project/Settings/Link
3. Colocar el archivo CLIPS.LIB en la carpeta C:\Archivos de programa\Microsoft Visual Studio\VC98\Lib
4. Añadir el siguiente archivo en la parte de Object/library modules
clips.lib

Comandos del *referee* y notificaciones

| | |
|---|---|
| COMM_HALT | 'H' |
| COMM_START | 's' |
| COMM_STOP | 'S' |
| COMM_READY | ' ' |
| COMM_FIRST_HALF | '1' |
| COMM_HALF | 'h' |
| COMM_SECOND_HALF | '2' |
| COMM_OVERTIME_HALF1 | 'o' |
| COMM_OVERTIME_HALF2 | 'O' |
| COMM_PENALTY_SHOOTOUT | 'a' |
| COMM_KICKOFF_BLUE | 'K' |
| COMM_KICKOFF_YELLOW | 'k' |
| COMM_PENALTY_BLUE | 'P' |
| COMM_PENALTY_YELLOW | 'p' |
| COMM_DIRECT_BLUE | 'F' |
| COMM_DIRECT_YELLOW | 'f' |
| COMM_INDIRECT_BLUE | 'I' |
| COMM_INDIRECT_YELLOW | 'i' |
| COMM_TIMEOUT_BLUE | 'T' |
| COMM_TIMEOUT_YELLOW | 't' |
| COMM_TIMEOUT_END_BLUE | 'Z' |
| COMM_TIMEOUT_END_YELLOW | 'z' |
| COMM_GOAL_BLUE | 'G' |
| COMM_GOAL_YELLOW | 'g' |
| COMM_GOAL_DEC_BLUE | 'D' |
| COMM_GOAL_DEC_YELLOW | 'd' |
| COMM_YELLOW_CARD_BLUE | 'Y' |
| COMM_YELLOW_CARD_YELLOW | 'y' |
| COMM_RED_CARD_BLUE | 'R' |
| COMM_RED_CARD_BLUE | 'r' |
| COMM_CANCEL | 'c' |

# Glosario

**BIOS.** Basic Input-Output System.
**CLIPS.** lenguaje utilizado para la programación de sistemas expertos.
**D1.** Rol del robot delantero 1.
**D2.** Rol del robot delantero 2.
**D3.** Rol del robot delantero 2.
**DLL.** Librerías de enlace dinámico.
**dribbler.** Sistema de los robots utilizado para retener la pelota.
**DSP.** Procesador Digital de Señales.
**EK.** Eagle Knights (equipo de robots del ITAM).
**encoder.** Señal cuadrada de los motores que varía su frecuencia dependiendo de su velocidad de movimiento.
**EKIntelSSL.** Sistema de IA del equipo Eagle Knights.
**EKVision.** Sistema de visión del equipo Eagle Knights.
**FIFA.** Federación Internacional de Futbol Asociación.
**GUI.** Interfaz Gráfica del Usuario.
**GET.** Árboles de Exploración Geométrica.
**HUB.** Concentrador *firewire.*
**IA.** Inteligencia Artificial.
**kicker.** Sistema de pateo utilizado en los robots.
**LARS.** Simposium Latinoamericano de Robótica.
**MFC.** Microsoft Foundations Classes.
**MLP.** Red neuronal de perceptrón multicapas.
**openGL.** Paquetería abierta de librerías gráficas.

**PID.** Filtro de control proporcional integral derivativo.
**PWM.** Señal modulada en el ancho del pulso.
**RRT.** Árboles aleatorios de rápida exploración.
**SDK.** Paquete para el desarrollo de software.
**SSL.** Liga de *RoboCup* "*Small Size League*".
**SPL.** Liga de *RoboCup* "*Standard Platform League*".
**thread.** Hilo de ejecución de un programa.
**timer.** Temporizador.
**transceiver.** Transmisor/receptor inalámbrico.
**VS.** Equipo de robots contrario.

# Bibliografía


[1] Elaine Rich, Kevin Knight, *Inteligencia artificial*, Alfaomega, México, 1994, caps: 4,21.

[2] José Santos Reyes, Richard J. Duro. *Evolución artificial y robótica autónoma*, México, 2005.

[3] Chris Nikolopoulos, *Expert systems: introduction to first and second generation and hybrid knowledge based systems*, New York, 1997.

[4] Joseph C. Giarratano, CLIPS: Users guide, Quicksilver Beta.

[5] Rodney A. Brooks. Cambrian Intelligence: the early history of the new AI. MIT Press, 1999.

[6] Kuhlmann, Federico y Antonio Alonso. Información y Telecomunicaciones. Fondo de Cultura Económica, México, 1996.

[7] LaValle, S. M. (1998). Rapidly-exploring random trees: A new tool for path planning. In Technical Report No. 98-11.

[8] The RoboCup Federation, 2007 results, "Small Size Robot League – robocup2006:results" . http://small-size.informatik.uni-bremen.de/robocup2008:schedule_and_results Último acceso: 11/11/08.]

[9] Oussama Khatib, "Real-Time Obstacle Avoidance for Manipulators and Mobile Robots". in International Journal of Robotics Research (1986) Stanford University, California.

[10] Torres Ernesto, Weitzenfeld Alfredo. "RoboCup Small-Size League: Using Neural Networks to Learn Color Segmentation during Visual Processing". In: LARS (2008), Salvador, Brasil.

[11] Joseph C. Giarratano, "Clips User's Guide", Quicksiver Beta, Dec. 2007.

[12] Martínez Gómez, Luis, *Sistema de Visión Para el Equipo de Robots Autónomos del ITAM*. Tesis (Ingeniero en Computación) ITAM, 2004.

[13] Moneo Soler, Francisco, *Sistema de Planeación de Alto Nivel para Robocup*. Tesis (Ingeniero en Telemática e Ingeniero en Computación) ITAM, 2005.

[14] Sotelo Iniesta, Edgar, *Diseño e Implementación de los Robots F180 del ITAM*. Tesis (Ingeniero en Telemática) ITAM, 2006.

[15] Soto Ruiz, Misael David, *Diseño experimental para el mejoramiento del subsistema de control de la pelota para los robots de la liga SmallSize*. Tesis (Ingeniero en Industrial) ITAM, 2007.

[16] Rodríguez Ordonez, Jesús,.*Diseño de la arquitectura y control de trayectorias para robots small size* Tesis (Ingeniero en Telemática) ITAM, 2008.

[17] The Kalman Filter. [http://www.cs.unc.edu/~welch/kalman/ Último acceso: 30/10/08.]

[18] CLIPS DLL http://ourworld.compuserve.com/homepages/marktoml/clipstuf.htm Último acceso: 08/11/08

[19] División Académica de Ingeniería, "ITAM - División Académica de Ingeniería", ITAM 2008. [http://dai.itam.mx/ Último acceso: 01/29/08]

[20] Sistemas de Comercio Electrónico, "ITAM - División Académica de Ingeniería", ITAM 2008. [http://dai.itam.mx/ Último acceso: 01/29/08]

[21] Robótica, "ITAM - División Académica de Ingeniería", ITAM 2008. [ http://www.cannes.itam.mx/Alfredo/Espaniol/Cursos/Robotica/Robotica.htm Último acceso: 08/31/08]

[22] Laboratorio de Robótica, "Eagle Knights ". [http://robotica.itam.mx/ Último acceso: 09/06/08].

[23] The RoboCup Federation, 2005 results, "Small Size Robot League – robocup2005:results" . [http://www.robocup.org/games/05Osaka/319.html Último acceso: 09/06/08.]

[24] The RoboCup Federation, 2006 results, "Small Size Robot League – robocup2006:results" . [http://small-size.informatik.uni-bremen.de/robocup2006:results Último acceso: 09/06/08.]

[25] The RoboCup Federation, 2007 results, "Small Size Robot League – robocup2006:results" . [http://small-size.informatik.uni-bremen.de/robocup2007:results Último acceso: 09/06/08.]

[26] The RoboCup Federation, *"What is RoboCup?"*. [ http://www.robocup.org/ Último acceso: 09/06/08.]

[27] The RoboCup Federation, Small-Size League, *"Small Size Robot League - start"*. [http://small-size.informatik.uni-bremen.de/ Último acceso: 09/06/08.]

[28] The RoboCup Federation, Small-Size League Rules, *"Small Size Robot League - rules"*. [http://small-size.informatik.uni-bremen.de/rules:main Último acceso: 09/06/08.]

[29] Torres Ernesto, Arquitectura SSL [http://www.cannes.itam.mx/Alfredo/Espaniol/Cursos/Robotica/Material/ArquitecturaSSL.doc]

[30] Rojas, Raúl. "Omnidirectional Control". Freie Universität Berlin, 2005

[31] Small Size Robot League Referee Box [http://small-size.informatik.uni-bremen.de/referee:start Último acceso: 09/06/08

[32] Rojas, Raúl. "Predicting away robot control latency", 2003

[33] Greg Welch, Gary Bishop "An Introduction to the Kalman Filter", (2006), University of North Carolina, EUA.

[34] Peter Jackson "*Introduction to Expert Systems*" Addison-Wesley, New York, 1999, caps:1,2

[35] Video de clasificación para la competencia de RoboCup Suzhou 2008 http://mx.youtube.com/watch?v=Ul6AeoXPDRw