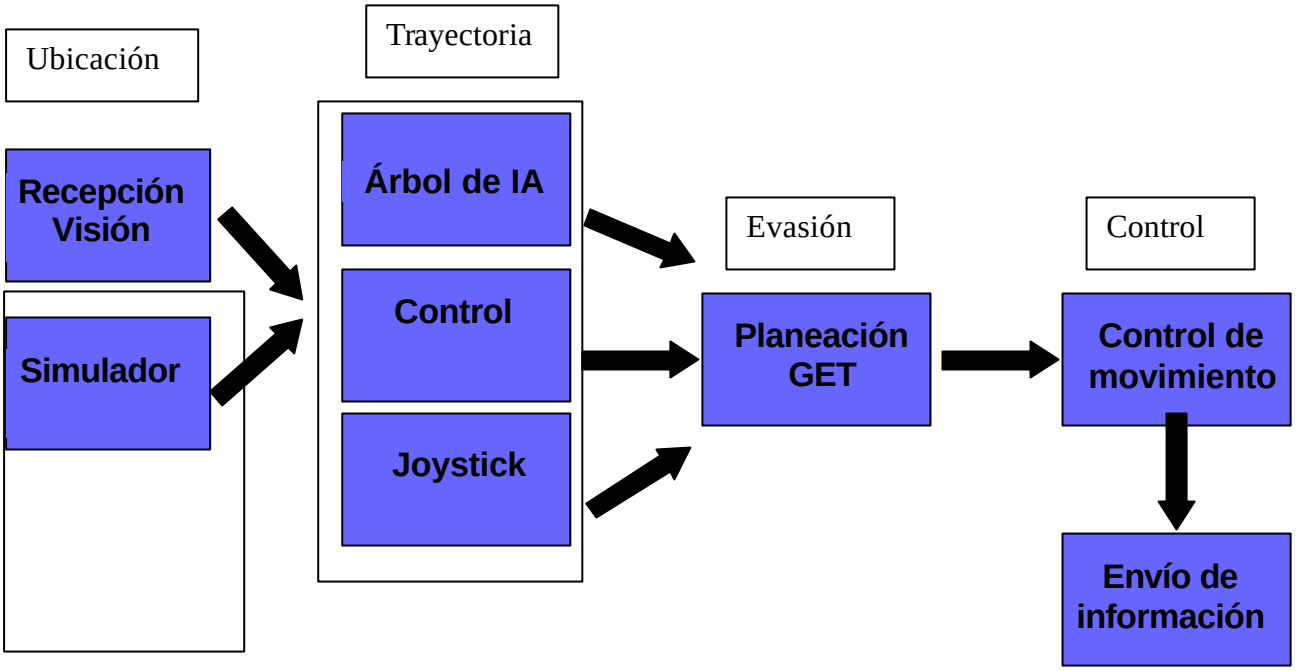
Ernesto Torres Vidal

# Sistema de Inteligencia Artificial del equipo Eagle Knights de la liga Small Size

## Introducción

La arquitectura del sistema completo consiste en un sistema de visión global, un sistema de inteligencia artificial para el control de los robots y un equipo de robots que reciben comandos por vía inalámbrica que definen los movimientos que deben realizar. La finalidad del sistema de inteligencia artificial es poder controlar al equipo formado por cinco robots de forma autónoma y realizar pruebas de las diferentes partes que componen al sistema. Para lograr esto es necesario poder probar los diferentes módulos de forma independiente. El sistema cuenta con diferentes modos de ejecución que permiten probar los diferentes módulos del sistema incluso cuando no se cuenta con la conexión con el sistema de visión, el referee o los robots. Para comprender la funcionalidad de los distintos módulos del sistema es necesario realizar una clasificación en dos partes. La primera consiste en como se define la ubicación de los objetos del campo de juego (robots y pelota) y la segunda es como se define la trayectoria que deseamos que realicen los robots. La ubicación se puede obtener de dos formas: por medio de un simulador que actualice la posición y la velocidad de los objetos con base en la dinámica de sus movimientos o por medio del sistema de visión que permite obtener la ubicación real de los objetos en el campo de juego. La trayectoria de los robots se puede definir de tres formas diferentes: la primera es por medio de un joystick, este dispositivo es útil para probar el control de los robots bajo diferentes velocidades, para conocer el tiempo de reacción y la precisión de los movimientos, la segunda forma es por medio de la interfase gráfica. Esta interfase permite generar trayectorias precisas variando la distancia, la dirección, el giro y la velocidad. Finalmente la trayectoria se puede definir por medio de un árbol de inteligencia artificial compuesto por diferentes comportamientos. Cada comportamiento define el movimiento que debe realizar el robot. Una vez que se ha definido una trayectoria se utiliza un planeador de rutas con la finalidad de evitar que los robots choquen entre sí. Además de estos modos el sistema cuenta con un ambiente gráfico que utiliza *OpenGL* para visualizar la forma en que el sistema está trabajando.




**Enradas y salidas del sistema**

El sistema de inteligencia artificial está diseñado para poder ejecutarse sin requerir forzosamente interconectarse con los demás módulos del sistema. Sin embargo cuando se desea que los robots jueguen de manera autónoma y se desean recibir comandos del árbitro desde una computadora neutral, como sucede en las competencias, es necesario realizar las conexiones. Además de recibir información del sistema de visión y del *Referee box* el sistema permite la conexión de otros dispositivos para la realización de pruebas como es el joystick y el uso del teclado y el ratón. Finalmente el sistema se debe comunicar con cinco o menos robots, para esto utiliza el puerto paralelo. La configuración técnica de estos dispositivos está descrita en la sección de ArquitecturaSSL.

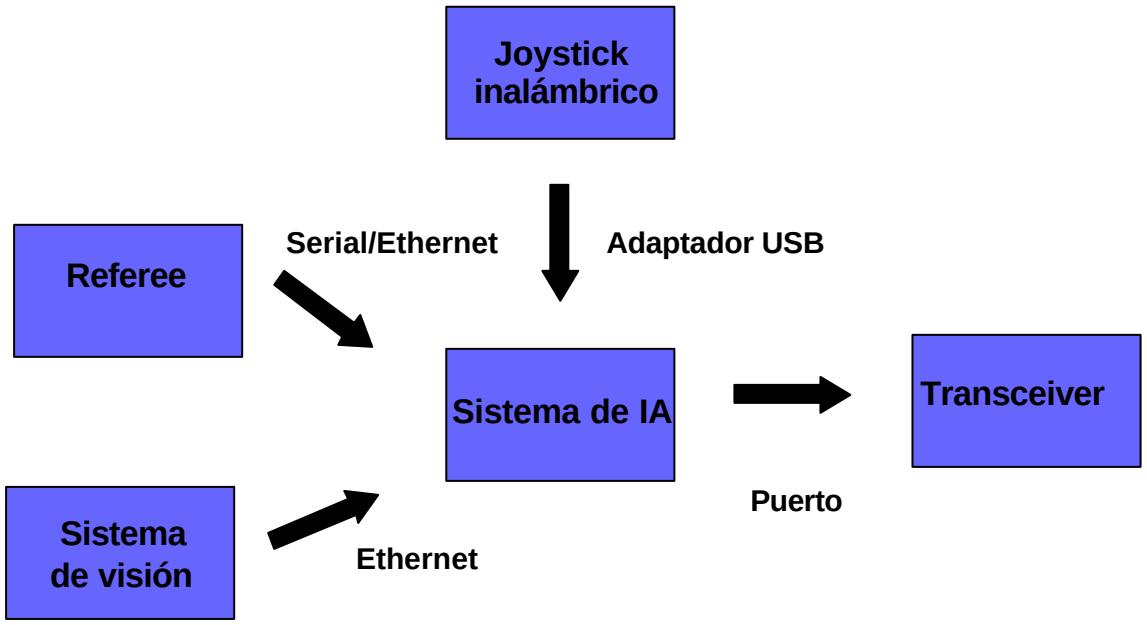


*Figura1. Conexiones del sistema de IA*

## Estructura de la aplicación

La aplicación está programada en el ambiente gráfico MFC con Visual C++. Está compuesto por un díalogo principal CEKIntelSSLDlg el cual se ejecuta al iniciarse la aplicación en CEKIntelSSLApp. Este díalogo principal contiene a una seria de diálogos desplegados dentro de un tabulador y a otro diálogo sobre el cual se dibuja el ambiente gráfico (CDlgCancha). Los nombres de los tabuladores y su diálogo asociado son:

| | |
|---|---|
| EK | CDlgRobots |
| VS | CDlgRobots |
| Int | CDlgIntel |
| IntVS | CDlgIntel |
| Sim | CDlgSimulador |
| Con | CDlgControl |
| Gra | CDlgGraficas |
| Ref | CDlgReferee |

El diálogo principal contiene además un el manejador principal de la aplicación (CManager). La ventaja de esta arquitectura es que cualquier diálogo puede obtener un apuntador al diálogo principal y de esta forma acceder a la información contenida en el mainManager.

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

Dentro del ciclo principal se realizan las siguientes funciones

1. Lectura de la información proveniente del sistema de visión.
2. Actualización de información en los diálogos.
3. Evaluación del árbol de inteligencia artificial.
4. Actualización de los eventos del joystick.
5. Cálculo de la trayectoria por el dialogo de control.
6. Control del robot.
7. Envío de información.

Este ciclo se está ejecutando constantemente. Es importante recalcar que el número de ciclos por segundos que puede ejecutar el sistema debe ser mayor al número de ciclos

por segundo al que trabaja el sistema de visión. De otra forma el sistema se alenta y se pierde el control de los robots.

## Lectura del escenario

La principal entrada al sistema proviene del sistema de visión. Este sistema entrega el escenario actual. Este escenario contiene toda la información que el sistema de visión ha podido obtener y consiste en las posiciones de los robots y de la pelota (milímetros), la orientación de nuestros robots (radianes), Además entrega la salida del filtro de kalman para la pelota (corrección y predicción) y la velocidad de cada uno de los objetos así como la velocidad de giro de nuestros robots (milímetros/segundo y radianes/segundo). Esta información se lee por medio de la estructura de datos Escenario por medio de un socket (clase CSocketUDP). Durante la recepción en la clase CRecepcionVision se copia esta información al escenario principal E (de la clase CEscenario) ya que además de la información proveniente del sistema de visión los robots y la pelota utilizan más información la cual está definida en las clases CRobotActual y CPelota.

## Inteligencia artificial

El módulo de inteligencia artificial es el núcleo del sistema. En este parte se definen las estrategias de alto nivel, en donde se incluye la coordinación entre robots y se define la forma en que se realizan los distintos comportamientos En esta parte se toma en cuenta el rol de cada robot para definir una estrategia independientemente de su número y se utiliza el estado de juego definido en la clase CGameControl.

Cada posible acción está definida dentro de un comportamiento (clase CComportamiento). Cada comportamiento debe definir la información contenida en el escenario ideal para ese momento (CSuperEscenario). La información que define es la posición a la que deseamos se mueva el robot, su orientación deseada, su velocidad y su velocidad de giro. Los comportamientos pueden tomar distintas complejidades. Un comportamiento muy sencillo como seguir a la pelota se encarga d definir la coordenada final igual a la pelota y definir los demás parámetros de movimiento. Es posible en este caso, por ejemplo, adecuar la velocidad dependiendo de que tan lejos se encuentre de la pelota para que el robot realice movimientos más suaves en ciertas condiciones. Otros comportamientos más complejos requieren la definición de diferentes estados. Un ejemplo es el tiro a gol. En este comportamiento se requieren tres etapas: En caso de que la pelota esté detrás del robot tiene moverse a un lado de la pelota, posterormente (y desde aquí se inicia en caso de que la pelota esté adelante) el robot se coloca detrás de la pelota y finalmente se dirige hacia la pelota y patea. Otros comportamientos como el zig-zag requieren que conocer el paso en el que se encuentra el primer robot. Para guardar el estado en el que se encuentra de forma ordenada se utilizan variables locales definidas como *static*. La ventaja de estas variables es que al terminar de ejecutar el método mantienen su valor, de tal forma que cuando se vuelve a ejecutar el mismo método en lugar de crearse una nueva variable local, se recupera el valor que tenía la última vez. Al programar la forma en que se realizan los comportamientos es indispensable tomar en cuenta el modelo de control dinámico de los robots ya que la aceleración y el frenado tanto de la velocidad lineal como de la velocidad de giro restringen muchas veces la forma en que se ejecutan los comportamientos.

Es importante que se plantee la estrategia de alto nivel una vez que se hayan resuelto los problemas básicos de control del robot y visión, de otra forma es imposible que la estrategia programada se realice correctamente en la realidad.

A continuación se muestra una lista con algunos comportamientos implementados:

CubreTiro
CubrePelota
MarcaKickoff
RecibeDespeje
BarreraPorteria
TiroGol
BloqueaTiro
LibreDespejePortero
BloqueaTiroArea
InterceptaPelota
CubrePorteria
CubrePorteriaPenal
SiguePelota
LineaRecta
Detener
PosicionPenal
PorteroPenal
LineaPenalNuestra
LineaPenalCont
TiroPenal
Tiro
TiroPelDetenida
Ocho
ZigZag
SigueAnterior

Ya que se han definido los comportamientos el siguiente paso es probarlos utilizando el simulador, esto permite resolver errores de programación que se verían reflejados en el movimiento del robot y que son más difíciles de corregir cuando hay otras variables que pueden estar afectando como es el sistema de visión o el hardware del robot. Una vez que funciona con el simulador se puede proceder a hacer los ajustes de velocidades y precisión utilizando la visión y uno o más robots dependiendo del comportamiento. Para la programación de los comportamientos se requiere contar con las dimensiones reales del campo de juego. Las dimensiones se definen en la clase CCancha y son las dimensiones totales de la cancha, la posición de las líneas, las dimensiones de las porterías, del área y del círculo central todo medido en milímetros. A continuación se muestra un diagrama con las dimensiones actuales del campo de juego:

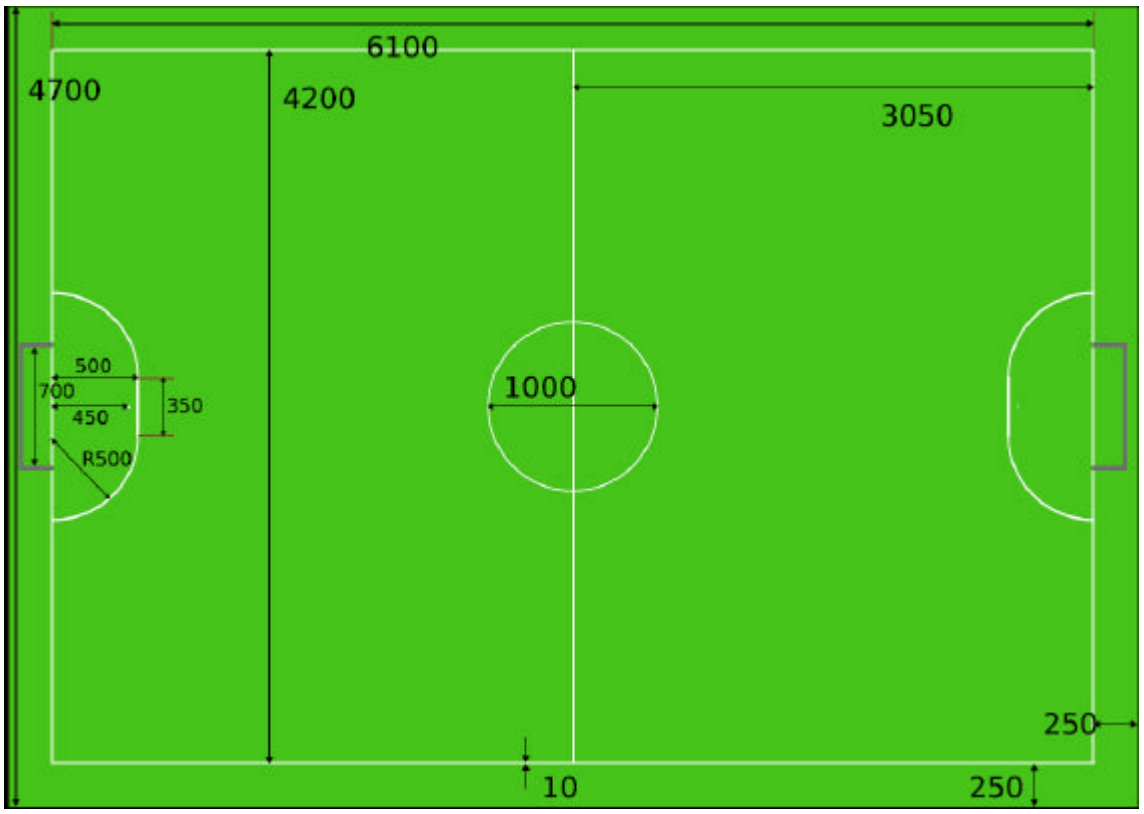


*Figura2. Dimensiones de la cancha*

Cada uno de los comportamientos funciona de forma independiente. El siguiente paso es definir la estrategia conjunta de todos los robots. Cada decisión que se tome se concluye con la ejecución de un comportamiento y los comportamientos que se definan para cada uno de los robots dependiendo de las circunstancias del juego deben estar coordinados entre sí para lograr que los robots jueguen en equipo. Por lo tanto se utilizan una serie de condiciones que permiten realizar la evaluación de lo que está sucediendo en el campo de juego.

Algunas de las condiciones implementadas son las siguientes (dentro de la clase CCondicion):

EdoJugada
CambioEdo
CondRecibeDespeje
CondInterceptaPelota
Numero
RolActivo
RolRegionTiro
RolPelota
PelotaDentroCancha
PelotaEntrePorteriaRobot
PelotaDelanteRobotX
PelotaAtrasRobotX
RolRelacionPelotaRobot

RolDelanteRobotX
AdelanteReferencia
RobotCercanoPelota
LibreFrente
LibreRol
DistanciaContrario
RolComportamiento
DistanciaPelota
DistanciaPelotaCentro
DistanciaPelotaPortEK
DistanciaPelotaPortVS
DistanciaContrarioPelota
RolRelacionMarcaCentral
PelotaAdelanteX
PelotaArribaY
PelotaAdelanteRobot
PelotaDetrasRobot
LibrePorteriaVS
LibrePorteriaEK
CondCubreTiro
CondMarcarCentralTiro
CondMarcarCentral
InterRadioPortNuestra
ContrarioAbajoY
ContrarioArribaY
DistanciaPortVS
DistanciaPortEK
RegionTiro
RegionPelotaPortVS
RegionPelotaPortEK
EdoJuego
EdoJuegoAnt
Rol

Ya que se tienen definidas las condiciones y los comportamientos se estructuran árboles que contienen la estrategia que van a seguir los robots. Se puede definir muchos árboles en donde cada uno plantea una estrategia diferente. Los roles utilizados son: Portero, Defensa, D1, D2 y D3. A continuación se muestra la rama del árbol derivada del estado de juego "Start" para el rol Portero. En esta estrategia el portero colabora con dos robots que le ayudan a defender la portería con los roles Defensa y D3. La lógica consiste en que en caso de haya uno o dos robot el portero pueda defender la portería sin dejar espacios libres. Se tiene que considerar que en los partidos es muy común que un robot tenga que salir del juego por problemas de hardware, bajo nivel de baterías, etc. La inteligencia artificial debe considerar todos los casos con el objetivo poder jugar si hay menos de cinco robots jugando.

Para este caso se utilizan las siguientes condiciones:

**Start**: Evalúa si el estado actual es "Start"
**Portero/PorteroNo**: Evalúa si el rol del robot que se está evaluando es "Portero"

**DentroAreaPortero/FueraAreaPortero**: Evalúa si la pelota está dentro de de un área medida a partir del centro de la portería.
**DefensaActivo/DefensaActivoNo**: Evaluá si el rol defensa está activo.
**D3Activo/D3ActivoNo**: Evalúa si el rol D3 está activo.
**DefensaRolBarreraPorteriaDer/DefensaRolBarreraPorteriaDerNo**: El rol defensa está cubriendo la portería desde la parte derecha.
**D3RolBarreraPorteriaIzq/ D3RolBarreraPorteriaIzqN**o: El rol D3 está cubriendo la portería desde la parte izquierda.

Independientemente de las condiciones utilizadas para escoger cual es el comportamiento adecuado dentro de cada comportamiento se realizan evaluaciones para decidir como realizar la acción.

Los comportamientos (hojas del árbol) utilizados son:

**DespejarPortero**: Despeja la pelota hacia los extremos de la media cancha.
**BloqueaTiroPortero**: Tiene dos casos: si la pelota se dirige hacia la portería (utilizando predicción de la pelota) o en caso de que no se cumpla esta condición. Si no se cumple se realiza una cobertura a partir del punto central (punto_cubrir) de la portería a cierta distancia.

```
coor_final.cartesiana(dist_cubrir,punto_cubrir.angulo_coordenadas(coor_pel));
coor_final=coor_final+punto_cubrir;
```

En caso de que la pelota se diriga hacia la portería se cubre a cierta distancia de la portería a partir del punto de intersección con la línea de gol. Se establecen dos límites con la finalidad de que el robot no se acerque tanto a los postes de la portería.

```
if(coor_inter.y>lim_sup)
      coor_inter.y=lim_sup;
else
      if(coor_inter.y<lim_inf)
            coor_inter.y=lim_inf;

coor_final.cartesiana(dist_cubrir,coor_inter.angulo_coordenadas(coor_pel));
coor_final=coor_final+coor_inter;
```

**BloqueaTiroPorteroLat**: Utiliza el comportamiento BloqueaTiroPortero. La diferencia es que el punto a cubrir en lugar de ser el punto medio de la portería cubre el poste más cercano a la pelota:

```
if(coor_pel.y>C->long_y/2.0)
punto_cubrir=CCoordenada(C->linea_izq,C->long_y/2.0+C>largo_porteria/3.0);
else
punto_cubrir =CCoordenada(C->linea_izq,C->long_y/2.0-C->largo_porteria/3.0);
```


*Figura3. Árbol de inteligencia artificial del portero*

El árbol completo se define en un archivo de texto. El formato de este archivo consiste en indicar el inicio y fin de un nivel (-1, -2, -3, -4, -5, etc) por cada uno de los nodos. Cada nodo tiene un número que facilita ver en que nivel se encuentra y el nombre del archivo asociado. Conforme se realiza la lectura de este archivo al inicio del programa se crea la estructura del árbol (utilizando la clase Arbol).

-1
1 Start
-2
111 Portero
-3
111 DentroAreaPortero
-4
1111 DespejarPortero
-4
111 FueraAreaPortero
-4
1111 DefensaActivo
-5
11111 D3Activo
-6
11111 DefensaRolBarreraPorteriaDer
-7
111111 D3RolBarreraPorteriaIzq
-8

1111111 BloqueaTiroPortero
-8
111111 D3RolBarreraPorteriaIzqNo
-8
1111111 BloqueaTiroPorteroLat
-7
-7
11111 DefensaRolBarreraPorteriaDerNo
-7
111111 BloqueaTiroPorteroLat
-7
-6
11111 D3ActivoNo
-6
111111 BloqueaTiroPorteroLat
-6
-5
1111 DefensaActivoNo
-5
11111 D3Activo
-6
111111 BloqueaTiroPorteroLat
-6
11111 D3ActivoNo
-6
111111 BloqueaTiroPortero
-6
-5
-4
-3
-2
-1

El árbol de condiciones utilizado en los roles Defensa y D3 consideran esta misma lógica y otras condiciones para seleccionar sus comportamientos.

En este caso se estructuró el árbol para que fuera binario (a partir de la condición Portero) ya que esto facilita comprender la lógica que se está siguiendo. Sin embargo por la forma en que se está evaluando el árbol es posible que el árbol tenga un grado mayor. Esto se puede ver al momento de evaluar el estado de juego:

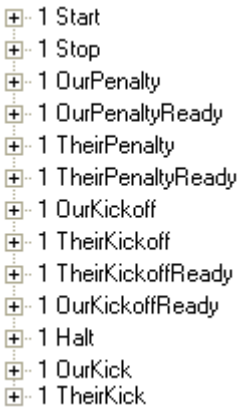



*Figura4. Primer nivel del árbol para decidir el estado de juego.*

En esta caso hay 13 posibles soluciones a la evaluación del primer nivel del árbol. La evaluación del árbol se basa en el método de búsqueda "Best First Search". Esta búsqueda consiste en ir evaluando cual nodo de los posibles para cada nivel es el que tiene mayor puntuación. Una vez que se ha determinado se selecciona este nodo y se repite el proceso para seleccionar cual de sus hijos es el que tiene mayor puntuación. Para determinar la puntuación cada una de las condiciones está ligada a un archivo de texto (dentro de la capeta JugadasEK) Los comportamientos no requieren tener condiciones asociadas sin embargo también tienen asociado un archivo de texto (vacío) ya que al momento de evaluar el árbol no se conoce si un nodo es condición o comportamiento hasta que el nodo no tenga más hijos. A continuación se muestra el caso de la condición **DetroAreaPortero** la cual se usa para decidir si el portero debe o no despejar la pelota:

1 0800DistanciaPelotaPortEK
11 Menor 1
12 Mayor 0
-1

La primera línea indica que es la primera función que se utiliza para evaluar la condición, recibe un solo parámetro (0800) que se utiliza como la distancia a evaluar el método llamado "DistanciaPelotaPortEK". En caso de que el resultado sea Menor el puntaje será 1 y si es Mayor el puntaje es 0. El -1 indica que se ha terminado de evaluar condiciones. El caso contrario **DentroAreaPorteroNo** tiene

Es posible que un nodo del árbol evalúa a más de una condición aunque se puede simplificar usando dos condiciones distintas agregando un nivel más al árbol lo cual agrega claridad y evita complicaciones.

La principal ventaja de estructurar de esta forma el árbol de inteligencia artificial es que una vez que se han programado los comportamientos y las condiciones se pueden estructurar diferentes estrategias sin tener que modificar el código. Es decir sólo hay que modificar el archivo que contiene el árbol y crear los archivos que definen las condiciones de sus nodos utilizando las condiciones de evaluación y los comportamientos ya programados.

Antes de realizar la evaluación del árbol se deben considerar la dirección en la que estamos tirando. Todos los comportamientos y condiciones consideran que se está tirando hacia la derecha. En el caso contrario es necesario invertir el escenario de nuestros robots con la finalidad de que la inteligencia calcule el equivalente de lo que deberían hacer si tiran hacia la derecha y una vez que se tiene el resultado se vuelve a invertir. Este proceso se realiza tanto para nuestro equipo como para el contrario para poder probar hasta 10 robots jugando 5 contra 5.

## Planeación de rutas

Los robots se encuentran dentro de un entorno que cambia constantemente en tiempo real. Para que los robots se puedan mover seguramente dentro del campo de juego deben plantear una ruta que los lleve correctamente a su destino. Los principales problemas con los que un planeador de rutas debe lidiar son los siguientes:

- El planeador debe ser eficiente para trabajar en tiempo real (30 cuadros por segundo).
- El espacio cambia constantemente.
- La configuración del espacio es impredecible.

Una forma de hacer esto es por medio de campos potenciales. Otra forma es utilizando árboles de exploración. Una solución para estos árboles que se utiliza frecuentemente son los RRT y su extensión ERRT. Un *rapidly randomized tree* consiste en un árbol que utiliza la aleatoreidad para explorar grandes espacios. Consiste en dos pasos: con probabilidad p se encuentra el punto más cercano al destino y se extiende en esa dirección, con probabilidad 1-p se escoge un punto aleatorio y se extiende hacia el. Este algoritmo se utilice frecuentemente en espacios con muchas dimensiones y entornos muy complicados. Sin embargo, nuestro entorno se conoce bien por lo que puede crecer usando esta información en lugar de explorar todo el espacio aleatoriamente. Los obstáculos pueden representarse como figuras geométricas para que el planeador pueda usar la forma de los objetos para encontrar una ruta eficiente dentro del espacio. Nos referimos a estos árboles como árboles de exploración geométrica (GET). Estos árboles son una forma eficiente de construir para que pueda usarse en cada iteración. Puede combinar distintos tipos de obstáculos. En nuestro caso los robots se representan como círculos y las porterías como rectángulos. La idea de un árbol GET es empezar extendiendo el árbol hacia su destino y evadir los obstáculos que le puedan interferir a su movimiento lineal.

Para la planeación de rutas el entorno se modela en un espacio bidimensional. La siguiente imagen muestra una escena de juego de la liga small size:

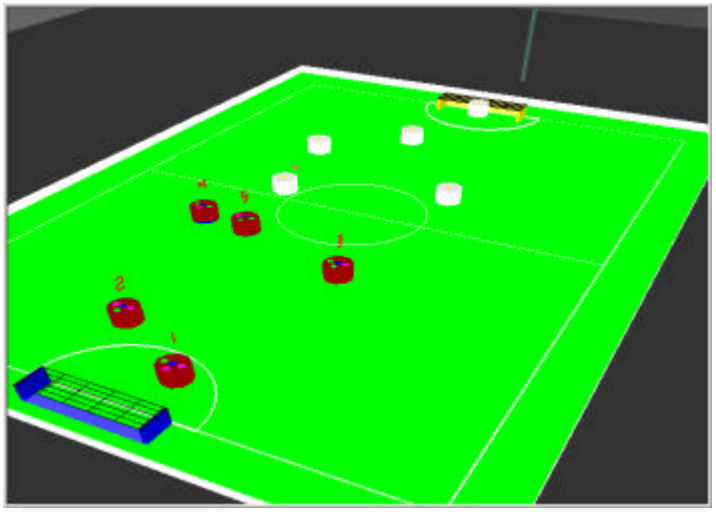


*Figura5. Escenario de jueg en 3 dimensiones.*

Para generar la planeación se define el punto inicial al igual que el punto final. Este último es entregado por el módulo de inteligencia artificial. El punto inicial se considera como un punto de exploración y se define como la raíz del árbol. En este ejemplo nos enfocaremos en el robot 1 el cual desea alcnazar la pelota. La siguiente imagen como se muestra a continuación:

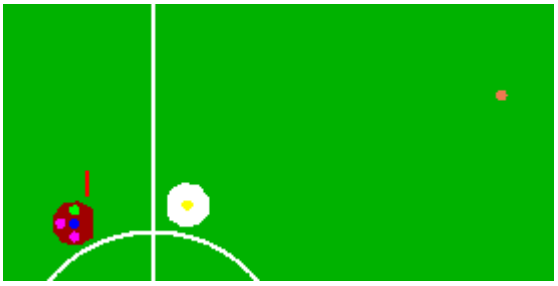


*Figura6. Planeación de rutas en un espacio de dos dimensiones.*

Los pasos que sigue el algoritmo son los siguientes:

Primero se selecciona un nodo de exploración del árbol y se intenta alcanzar la meta avanzando una pequeña distancia predefinida. Si no hay obstáculos interfiriendo con el nuevo punto entonces el árbol se extiende hacia el nuevo punto que se convierte en el nuevo punto de exploración. Para saber si un obstáculo interfiere entre el punto inicial y el final se debe considerar la geometría del obstáculo. El vector A se defino desde el punto inicial hacia la meta. En el caso de un obstáculos circular como otro robot se calcula una posible intersección entre el círculo de radio R y el vector A. Esta intersección se calcula resolviendo una ecuación de segundo grado. La siguiente imagen muestra los puntos de intersección:

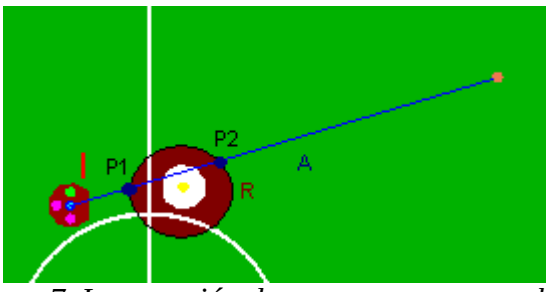
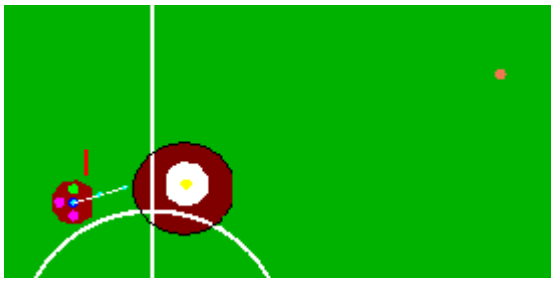
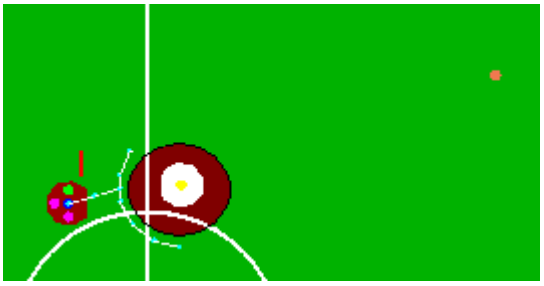



*Figura7. Intersección de un vector con un robot*.

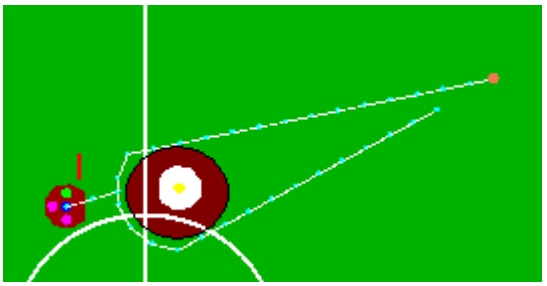
En este ejemplo se tienen dos intersecciones: P1 y P2. Mientras el nodo de exploración esté fuera del un radio R se sigue extendiendo hacia la meta. Cuando el árbol alcanza el radio del obstáculo el árbol genera dos posibles rutas una hacia cada lado del obstaculo. Ahora se cuenta con dos nodos de exploración. Mientras los nodos de exploración no puedan alcanzar libremente la meta se continúa rodeando el obstáculo hasta que no exista interferencia o se encuentre otro obstáculo. Las siguientes imágenes muestran el proceso de generación del árbol:



**Paso 1**



**Paso2**



**Paso 3**

*Figura8. Generación del árbol.*

Una vez que el árbol ha alcanzado un punto lo suficientemente cercano a la meta se elige la trayectoria más corta y el robot se envía directamente al primer punto antes de la intersección o hacia la dirección del siguiente nodo hacia la meta. La evaluación del algoritmo en el tiempo se muestra a continuación:

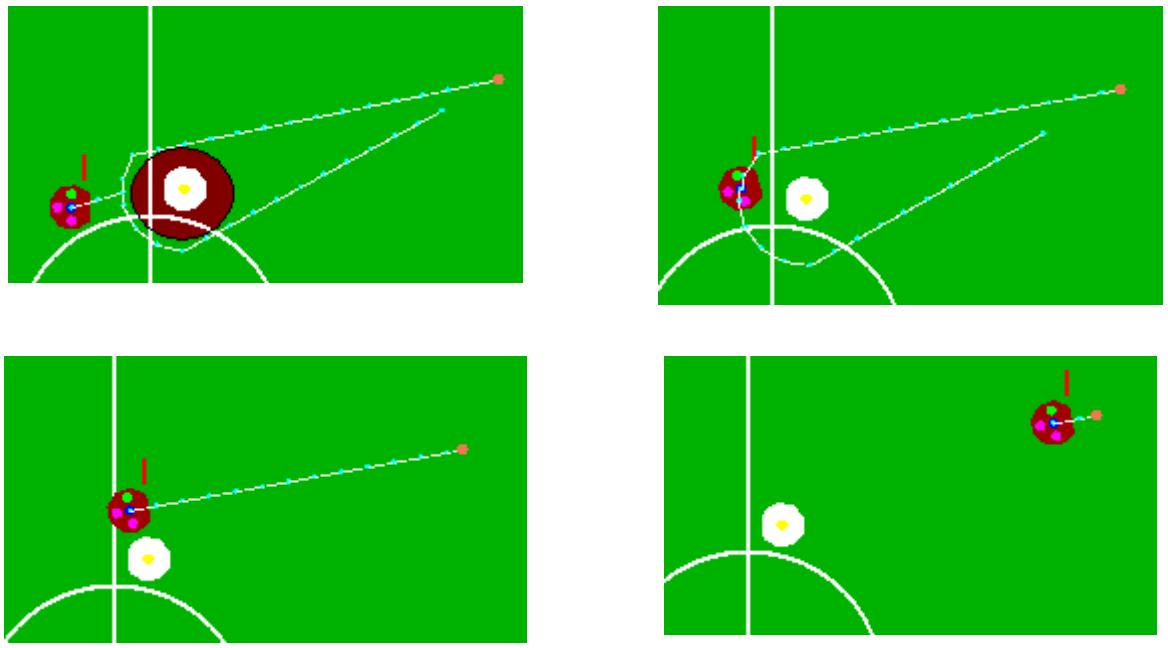
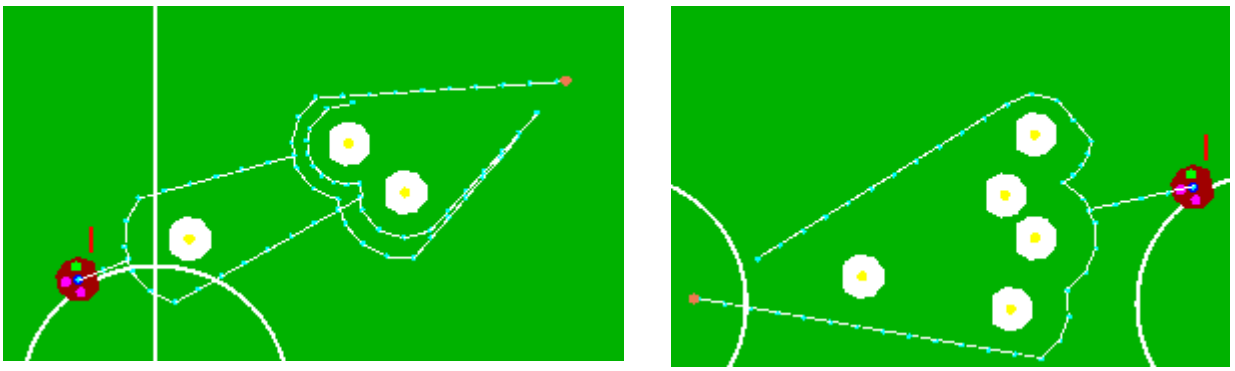


*Figura9. Planeación de rutas en el tiempo.*

En caso de que un nodo de exploración esté rodeando un obstáculo y durante este rodea se encuentre con un nuevo obstáculo se continúa con la evasión en la misma dirección pero se toma como referencia el nuevo obstáculo. Todos estos puntos se generan tangencialmente al obstáculo. Otros obstáculos como las porterías se pueden representar como rectángulos. A continuación se muestran casos más complicados y como los resuleve este algoritmo:



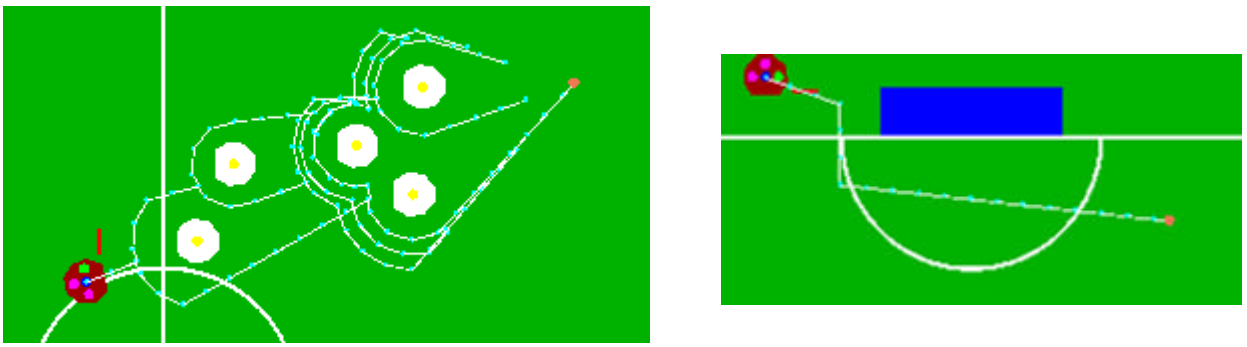


*Figura10. Evasión de varios obstáculos juntos.*

Este algoritmo trabaja lo suficientemente rápido para trabajar en tiempo real. En cada ciclo de control se genera el árbol de un robot de tal forma que si están los cinco robots jugando cada árbol se mantiene por cinco cuadros. La principal dificultad de la implementación del algoritmo fue resolver el caso en el que en el tiempo se tengan dos rutas con una distancia similar y se tiene que escoger y mantener la misma elección para evitar que el robot dude entre las dos rutas.