Proyecto FIbRA

Estado del arte

Gustavo Armagno
garmagno@fing.edu.uy

Facundo Benavides
fbenavid@fing.edu.uy

Claudia Rostagnol
crostagnol@gmail.com

Tutores
Gonzalo Tejera, Ernesto Copello

http://www.fing.edu.uy/inco/grupos/mina/pGrado/FIRA2005.html

26 de noviembre de 2006

Instituto de Computación
Facultad de Ingeniería - Universidad de la República
Montevideo - Uruguay

Resumen

Este documento presenta el estado del arte del fútbol de robots y las técnicas y herramientas encontradas para resolver problemas como aprendizaje, comportamiento, *path planning*, colaboración entre jugadores, selección de estrategias de juego, etc. Asimismo, incluye una breve descripción de los aspectos relevantes de algunos equipos de fútbol de robots, como FRUTo, UBASot y PATITO FC.

En las siguientes secciones se presentan algunos conceptos relacionados con Agentes Inteligentes y algunas herramientas utilizadas para implementar sistemas de aprendizaje. Además se describen las características y conceptos utilizados en Algoritmos Evolutivos, profundizando el análisis para el caso de la Programación Genética. Se realiza una introducción al concepto de Sistemas Multiagentes y se desarrollan algunos conceptos asociados con el paradigma de las relaciones sociales en grupos de animales, trasladado a sociedades de agentes.

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción

El fútbol de robots ha sido un área de amplio crecimiento en los últimos años y ofrece un marco teórico adecuado para el estudio y aplicación de algunas técnicas y herramientas tales como construcción de robots, visión de robots, inteligencia artificial, cooperación entre agentes y procesamiento en tiempo real.

Este documento presenta el estado del arte del fútbol de robots y las técnicas y herramientas encontradas para resolver problemas como aprendizaje, comportamiento, *path planning*, colaboración entre jugadores, selección de estrategias de juego, etc. Asimismo, incluye una breve descripción de los aspectos relevantes de algunos equipos de fútbol de robots, como FRUTo, UBASot y PATITO FC.

El documento se organiza en varias secciones que abarcan temas que contribuyen al desarrollo de un equipo competitivo de SimuroSot.

El capítulo 2 presenta algunos conceptos relacionados con Agentes Inteligentes y algunas herramientas utilizadas para implementar sistemas de aprendizaje, como Redes Neuronales Artificiales y Aprendizaje por Refuerzo. Se describe también un algoritmo de Aprendizaje por Refuerzo en espacios de estados continuos.

El capítulo 4 describe las características y conceptos utilizados en Algoritmos Evolutivos, profundizando el análisis para el caso de la Programación Genética.

En la sección 3 se introduce el concepto de Sistemas Multi-agentes (*Multi Agent System - MAS*), y se describen las distintas formas de comunicación y cooperación que se pueden utilizar para modelar la interacción entre agentes de un sistema *MAS*. Esto da lugar a las ideas presentadas en el capítulo 5, donde los *MAS* son modelados como Sociedades de Agentes. Allí se presenta el paradigma de las relaciones sociales en grupos de animales, trasladado a sociedades de agentes que deben interactuar entre sí para lograr un objetivo común.

Una vez analizados todos estos aspectos, el capítulo 7 presenta las características de diseño e implementación de tres equipos de fútbol de robots que han competido en la categoría SimuroSot de la FIRA. Dos de estos grupos fueron desarrollados en Argentina, mientras que el tercero fue desarrollado en Uruguay. El equipo UBASot, desarrollado en la Universidad de Buenos Aires en el marco de un proyecto de fin de carrera, compitió a nivel mundial en el año 2002, obteniendo el 3^{er}puesto en su categoría. El equipo Argentino PATITO FC obtuvo el 1^{er} premio en su categoría en el campeonato Argentino (CAFR) del año 2004. También se presenta el equipo FRUTo, el cual ha sido desarrollado en la Facultad de Ingeniería durante el año 2003 y es la referencia más cercana en el área de fútbol de robots.

Capítulo 2

Agentes Inteligentes

A continuación, se verán algunas definiciones de agente, relacionadas con Inteligencia Artificial, extraídas de diferentes autores y presentadas en [FG96].

- *Un agente es cualquier cosa que puede percibir su entorno a través de sensores y actuar en base a dicho entorno a través de efectores* (extraído de "Artificial Intelligence: A Modern Approach" de Russell and Norvig, 1995).
- *Los agentes autónomos son sistemas de computación que habitan determinado ambiente complejo y dinámico, sensan y actúan autónomamente en este ambiente, y a través de sus hechos llevan a cabo las tareas que se les asignan* (definición presentada por Maes en 1995).
- *Permítanos definir un agente como una entidad de software persistente dedicada a un propósito específico. 'Persistente' diferencia un agente de una subrutina: los agentes tienen sus propias ideas sobre cómo desempeñar una tarea, sus propias agendas. 'Propósito específico' diferencia un agente de una aplicación multifunción: un agente generalmente es mucho más pequeño* (definido por Smith, Cypher y Spohrer en 1994).
- *Los agentes inteligentes llevan a cabo tres tareas continuas: percepción de condiciones dinámicas en el ambiente; acciones que afectan las condiciones del ambiente; y razonan para interpretar percepciones, resolver problemas, hacer inferencias y determinar acciones* (presentada por Hayes-Roth en 1995).
- *Un agente es un sistema de computación que se encuentra situado en determinado ambiente, y es capaz de actuar autónomamente en dicho ambiente con el fin de alcanzar sus objetivos de diseño. ...un sistema de computación de hardware o (más usualmente) software-based posee las siguientes propiedades* (definido por Wooldridge y Jennings en 1995)*:*
  - *autonomía: los agentes operan sin la intervención directa de humanos y tienen cierto tipo de control sobre sus acciones y estados internos;*
  - *habilidad social: los agentes interactúan con otros agentes (y posiblemente con humanos) a través de algún tipo de lenguaje de comunicación entre agentes;*
  - *reacción: los agentes perciben su ambiente (el cual puede ser el mundo físico, un usuario a través de una interfaz gráfica, una colección de otros agentes, la Internet, o tal vez todas estas combinadas), y responde en un tiempo razonable para cambiar lo que sucede en él;*
  - *pro-activos: los agentes no solamente actúan en respuesta a su ambiente, son capaces de exhibir comportamientos alineados con un objetivo tomando iniciativas.*

Por otro lado, Bittencourt [Bit05] presenta la siguiente definición de agente: *Se llama agente a una entidad real o abstracta que es capaz de actuar sobre ella misma o sobre su ambiente, que dispone de una representación parcial de este ambiente, que, en un universo multi-agente, puede comunicarse con otros agentes, y cuyo comportamiento es consecuencia de sus observaciones, de su conocimiento y de las interacciones con otros agentes.*

2.1. Clasificación de agentes

A grandes rasgos, los agentes pueden ser clasifican en tres grandes grupos:

- reactivos
- deliberativos
- híbridos

2.1.1. Reactivos

Se basan en el modelo condición-acción[1]. Se limitan a percibir el ambiente y en base al estado del mismo definen una acción. En ambientes que son totalmente observables pueden desempeñarse correctamente pero están expuestos a *deadlocks* o *livelocks* en ambientes parcialmente observables. Los mejores resultados se han obtenido en ambientes estables que no sufren modificaciones considerables en forma repentina. [Bit05]

Esta aproximación se destaca por la rapidez que le imprimen a la toma de decisiones siendo ésta su principal fortaleza. Por el contrario, los comportamientos logrados suelen ser simples y, al no tener memoria ni capacidad de retener estados anteriores, son incapaces de adaptar sus decisiones a los cambios.

Habitualmente son construidos en base a máquinas de estado o árboles de decisión. Son relativamente simples de construir pero presentan serios problemas prácticos de inflexibilidad y capacidad de adaptarse a cambios en la implementación. En los sistemas que adoptan este enfoque todas las decisiones deben estar predefinidas de antemano, incluyendo la coordinación entre agentes. Estas restricciones afectan sobre todo a los sistemas multi-agentes donde es importante que cada uno pueda anticipar las consecuencias de sus actos y la de otros. Los resultados, desde el punto de vista de la cooperación, no son satisfactorios. El comportamiento grupal observado en la mayoría de los casos se parece más a un cúmulo de agentes persiguiendo objetivos individuales que a un equipo con objetivos globales.

En conclusión, se verifican comportamientos bastante primitivos con los cuales es difícil lograr objetivos no triviales en ambientes complejos, dinámicos e incompletos. [PAC04]

2.1.2. Deliberativos

Los sistemas deliberativos, a diferencia de los reactivos, basan sus decisiones en el modelo condición-cognición-acción. Este ciclo agrega una etapa donde, clásicamente, se realiza reconocimiento de la información del entorno y planificación. Suelen considerar información del pasado reciente además de la que reciben a través de los sensores. Pueden considerar predicción para anticipar el efecto de acciones pasadas y también el uso de motores lógicos que permitan mejorar el manejo de reglas y proposiciones que se utilizan para modelar los diferentes escenarios. [PAC04]

[1] En el modelo condición-acción las acciones son elegidas exclusivamente a partir de reglas simples que mapean percepciones en el conjunto de acciones posibles.

Entre las ventajas que presentan sobre los sistemas reactivos se destaca el poder de expresividad, entendido éste como la capacidad de modelar aspectos del mundo que los sistemas reactivos no tienen.

Una debilidad importante de este enfoque es el tiempo de cómputo que puede requerir adoptar una única decisión. Esta característica no debe subestimarse en general, y particularmente en sistemas de tiempo real o con fuertes restricciones en el tiempo de respuesta. Dependiendo de la cantidad de aspectos del mundo tomados en cuenta y cuán complejo sea el análisis realizado, una sola decisión podría tomar desde una fracción de segundo hasta minutos, tornando este enfoque prohibitivo en algunos entornos como el fútbol de robots.

2.1.3. Híbridos

Los enfoques reactivo y deliberativo muestran fortalezas y debilidades. Este enfoque intenta ser una solución que adopta lo mejor de ambos minimizando sus debilidades. La propuesta de la aproximación híbrida es tomar la expresividad de la aproximación deliberativa y el tiempo de procesamiento de la aproximación reactiva, intentando tomar decisiones inteligentes garantizando el tiempo de respuesta. Para ello los sistemas híbridos se conforman, básicamente, de dos componentes de toma de decisiones: uno deliberativo y otro reactivo. Estos componentes ejecutan en paralelo todo el tiempo. Normalmente, el sistema utiliza las decisiones adoptadas por el componente deliberativo, pero si la respuesta toma demasiado tiempo utiliza la componente reactiva. La premisa es: la decisión reactiva no es tan buena como la deliberativa pero es mejor que no hacer nada. [PAC04]

2.2. Tipos de agentes

A su vez, los agentes también pueden ser divididos en cinco tipos, creando una jerarquía de agentes, donde cada nivel implica un mayor grado de inteligencia:

- agentes reactivos simples
- agentes reactivos basados en modelo
- agentes basados en metas
- agentes basados en utilidad
- agentes aprendices

2.2.1. Agentes reactivos simples

Se basan en un modelo condición-acción determinado por reglas de acción definidas para cada condición posible del ambiente. No tienen memoria de los hechos y generalmente se mueven en ambientes pequeños. Se limitan a percibir el ambiente y, en base al estado del mismo, realizan alguna acción. Pueden comportarse relativamente bien en ambientes totalmente observables, sin embargo, pueden caer en *deadlocks* o *loops* infinitos si el ambiente es observado parcialmente en cada momento, ya que no cuentan con memoria ni capacidad de retener estados anteriores para poder construir una imagen total del ambiente.

Son utilizados en ambientes estables que no sufren modificaciones considerables a lo largo del tiempo, ya que estos agentes no son capaces de retener internamente dichos cambios y adaptar sus acciones a los mismos.

2.2.2. Agentes reactivos basados en modelo

Cuentan con un modelo del ambiente que les permite conocer los diferentes estados en los cuales dicho ambiente puede encontrarse en cada momento. En base a esos estados, y las reglas de acción definidas para cada uno de ellos, los agentes toman decisiones y actúan. Sin embargo, no son capaces de encadenar reglas para obtener resultados de largo plazo, por lo tanto no tienen objetivos o metas. Solo pueden tomar decisiones en base al estado actual, las reglas que conocen y los posibles estados (solo un paso hacia adelante) a los que pueden llegar con cada acción.

Si bien los agentes basados en modelo pueden ser eficientes al momento de tomar decisiones, la incapacidad de planificación a largo plazo puede volverlos poco útiles en sistemas donde se requiere que cada agente sea "consciente" de sus propias metas.

Este tipo de agentes se desempeña mejor en ambientes pequeños y determinísticos.

2.2.3. Agentes basados en metas

Además de contar con un modelo interno del ambiente y reglas que utilizan para tomar decisiones en el mismo, pueden establecer objetivos o metas de largo plazo, ya que internamente pueden proyectar el estado futuro del ambiente en caso de tomar determinada decisión o acción.

Este tipo de agentes puede desenvolverse en ambientes determinísticos grandes y/o complejos.

2.2.4. Agentes basados en utilidades

Los agentes basados en utilidades son más adecuados para ambientes no determinísticos. Generalmente poseen un modelo interno en el cual se asigna una utilidad o "preferencia" a cada acción a partir de cada estado. De esta forma el agente tomará la decisión que le brinde mayor utilidad según el estado en el que se encuentre.

Se incorpora el concepto de **función de utilidad** para establecer la preferencia de un agente o sistema por un estado determinado. Esta función de utilidad retorna un valor para cada estado, el cual indica qué tan deseable es éste para el agente o el sistema.

Este tipo de agentes no tiene restricción en cuanto al ambiente en el que se desenvuelve.

2.2.5. Agentes aprendices

Son adaptables y tienen la capacidad de aprender y modificar sus creencias para modificar así luego sus decisiones o acciones futuras. Generalmente, se componen por varios elementos, entre ellos un componente de toma de decisiones y ejecución de acciones, y otro de aprendizaje, que modifica las creencias o conocimiento del agente para afectar así, indirectamente, la toma de decisiones futuras. Estos agentes también pueden contar con componentes para generar nuevos problemas o situaciones a los que deben enfrentarse para que puedan aprender y explorar nuevos estados, decisiones y acciones.

Este tipo de agentes no tiene restricción en cuanto al ambiente en el que se desenvuelve.

Más formalmente, un agente con capacidad para aprender puede ser dividido en cuatro componentes conceptuales, los cuales se muestran en la figura 2.1 y se detallan a continuación.

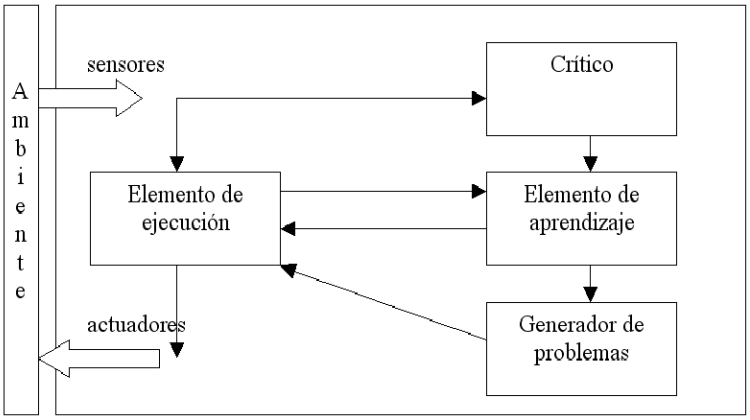


Figura 2.1: Modelo de un agente con capacidad de aprender.

Las mejoras en cuanto al conocimiento y la toma de decisiones es llevado a cabo por el **elemento de aprendizaje**. Por otro lado, las acciones a ejecutar son elegidas por el **elemento de ejecución**, quien recibe las percepciones del entorno y decide que acción tomar. El elemento de aprendizaje se retro-alimenta a partir de su conocimiento y del resultado obtenido de la ejecución de una acción elegida por el elemento de ejecución. En base a dicha retroalimentación decide cómo debe modificarse el elemento de ejecución o la información que éste maneja, para lograr una mejor toma de decisiones futura. Un tercer componente es el **crítico**. Este elemento monitorea el elemento de ejecución e informa al elemento de aprendizaje la evaluación realizada.

El cuarto y último componente típico de un agente con capacidad de aprender es el **elemento generador de problemas**. Este componente propone nuevos problemas o tareas (combinación de acciones) a ser probadas y exploradas por el agente con el fin de aprender nuevas estrategias, explorar lugares desconocidos, mejorar su comportamiento, etc.

2.3. Enfoques en Inteligencia Artificial

En Inteligencia Artificial y su relación con los agentes inteligentes existen básicamente dos enfoques:

- Inteligencia artificial centralizada.
- Inteligencia artificial distribuida.

El primer enfoque habla de agentes inteligentes donde toda la inteligencia del sistema se encapsula en un único componente.

A diferencia de esto, en el segundo enfoque, la inteligencia del sistema se encuentra distribuida en varias entidades. Dentro de este enfoque, a su vez, encontramos dos líneas de investigación. Por un lado se encuentra el estudio de comportamientos emergentes, donde de la unión de comportamientos y decisiones simples surgen comportamientos complejos. Esta línea de investigación recibe aportes del estudio de sociedades de insectos (hormigas, abejas, etc), donde se constatan comportamientos complejos, que podrían llamarse inteligentes, a pesar de que estos individuos no se consideran inteligentes, ya que su comportamiento individual esta "grabado" en el ADN. Una característica importante de este tipo de sistemas, es el no determinismo del comportamiento del mismo, dado que no se conoce *a priori* la evolución del colectivo. Generalmente los resultados se miden en base a la observación.

Por otro lado se encuentran las comunidades de agentes inteligentes, donde cada uno de los agentes del sistema, además de contar con cierta inteligencia para la toma de decisiones individuales, posee habilidades sociales tales como la colaboración, la cooperación o la competencia. Esta línea de investigación se encuentra actualmente en expansión.

Pueden ser utilizadas diferentes técnicas de inteligencia artificial para proveer de inteligencia a un agente, incluso pueden combinarse varias de ellas en un mismo agente para resolver distintos problemas o complementarse unas con otras.

En las siguientes subsecciones se describen algunas técnicas utilizadas para implementar agentes inteligentes.

2.4. Aprendizaje en redes neuronales

2.4.1. Composición de las redes neuronales

De acuerdo a Ben Krose y Patrick van der Smagt [KvdS96], una red neuronal artificial (*Artificial Neural Network - ANN*) está compuesta por un conjunto de unidades de procesamiento las cuales se comunican entre sí enviándose señales a través de un gran número de conexiones con un peso asociado. Como complemento, James A. Freeman y David M. Skapura [FS91] hacen énfasis en la estructura de la *ANN*, definiéndola como una colección de *procesadores paralelos* conectados entre sí formando un grafo dirigido, organizado de forma tal que la estructura misma de la red se adapte al problema considerado.

Del conjunto de conceptos asociados al modelo de procesamiento paralelo distribuido (*Parallel Distributed Processing - PDP*) se distinguen los siguientes componentes de la *ANN* (ver figura 2.2):

- Un conjunto de nodos, unidades o elementos de procesamiento. Todos estos términos serán utilizados indistintamente a lo largo de esta sección.
- Un estado de activación y_k para cada nodo, el cual es equivalente a la salida del mismo.
- Conexiones entre los nodos. Generalmente cada conexión posee un peso w_{jk} -denominado también fuerza o fortaleza de conexión (del inglés *connection strength*)- el cual determina el efecto que la señal del nodo j tiene sobre el nodo k.
- Una regla de propagación, la cual determina la entrada efectiva s_k de un nodo a partir de sus entradas externas.
- Una función de activación $\mathcal{F}_k$, la cual determina el nuevo nivel de activación a partir de la entrada efectiva $s_k(t)$ y la activación actual $y_k(t)$.
- Una entrada externa θ_k para cada nodo.
- Un método para obtener información, también conocido como regla de aprendizaje.
- Un entorno dentro del cual el sistema debe operar. El entorno provee señales de entrada y, en caso de ser necesario, señales de error.

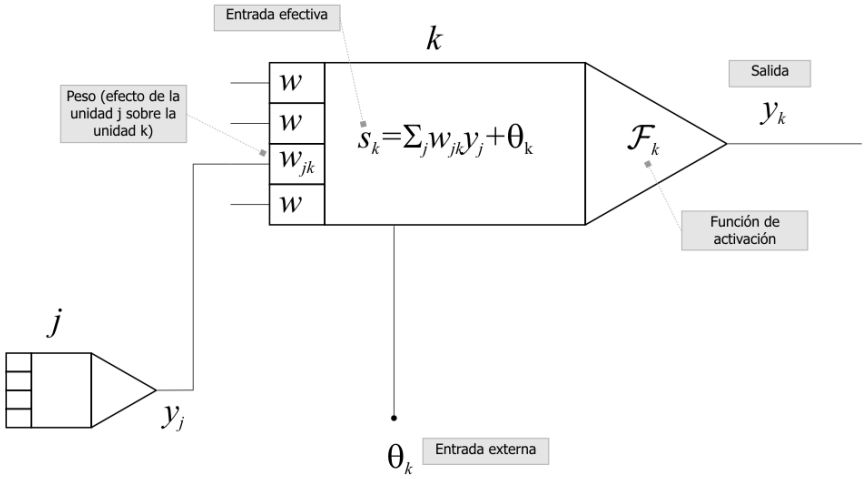


Figura 2.2: Componentes básicos de una red neuronal artificial. En este caso, la regla de propagación utilizada es la sumatoria de pesos.

Nodos

Todos los nodos reciben una entrada, la procesan y generan una salida. Algunos nodos -denominados *nodos de entrada*- se encuentran conectados al ambiente exterior para poder interactuar con el mismo. Estos nodos reciben la entrada del exterior y no de otros nodos. Por otro lado, algunos nodos -denominados *nodos de salida*- reciben la entrada de algunos nodos de la red pero producen su salida hacia el exterior. Los nodos que sólo reciben entradas de nodos de la red y transmiten su salida a otros nodos de la red se conocen como *nodos internos* u *ocultos*.

Los cálculos se realizan localmente en cada unidad o nodo, sin que sea necesario realizar cálculos a nivel de todo el sistema o conjunto de nodos. Además, el sistema es inherentemente paralelo, en el sentido de que muchos nodos pueden realizar cálculos al mismo tiempo. Estos cálculos locales dan lugar a posibles variaciones en los pesos de cada conexión y así determinar el comportamiento del sistema global.

Durante la operación, los nodos pueden ser actualizados *sincrónicamente* o *asincrónicamente*. En la actualización sincrónica, todos los nodos actualizan su activación en forma simultánea. En la actualización asincrónica, cada nodo posee una probabilidad de actualizar su activación en el tiempo t y, en general, sólo puede hacerlo un nodo a la vez.

Conexiones entre los nodos

En la mayoría de los casos, se asume que la contribución de un nodo en la entrada de otro nodo es aditiva. La *entrada efectiva* del nodo k es la suma ponderada de las señales incidentes (provenientes de otros nodos o del exterior) más la entrada externa θ_k:

$$s_k(t) = \sum_j w_{jk}(t)\, y_j(t) + \theta_k(t) \tag{2.1}$$

Se denomina *excitación* a la contribución de una entrada positiva e *inhibición* a la contribución de una entrada negativa. Aquellos nodos que poseen la regla de propagación 2.1 son llamados *nodos sigma*.

Modelo de activación

A partir de la entrada efectiva, se calcula el *valor de activación*, o simplemente la activación del nodo. La dependencia entre la activación del nodo y la entrada efectiva está determinada por una función $\mathcal{F}_k$ que toma la entrada efectiva $s_k(t)$ y la activación actual $y_k(t)$ y produce un nuevo valor de activación para el nodo k:

$$y_k(t+1) = \mathcal{F}_k(y_k(t),\, s_k(t)) \tag{2.2}$$

Como muestra la fórmula 2.2, la activación actual $y_k(t+1)$ puede depender del valor de activación anterior $y_k(t)$. Sin embargo, a menudo la función de activación es una función no-decreciente de la entrada total del nodo:

$$y_k(t+1) = \mathcal{F}_k(s_k(t)) = \mathcal{F}_k\left(\sum_j w_{jk}(t)\, y_j(t) + \theta_k(t)\right) \tag{2.3}$$

En general, la función de activación es modelada por una *función umbral* o *función de transferencia*, que obliga a que la entrada efectiva supere un determinado valor para que el nodo se active. Como muestra la figura 2.3, se utilizan funciones de tipo *escalón* (por ejemplo, la función *sgn*), funciones *lineales* o *semi-lineales*, o funciones de *saturación* (más suaves que las anteriores). Para estas últimas se utilizan comúnmente funciones de tipo *sigmoidea* (con forma de S), como:

$$y_k = \mathcal{F}_k(s_k) = \frac{1}{1 + e^{-s_k}} \tag{2.4}$$

En algunas aplicaciones se utiliza la función tangencial hiperbólica, que produce valores de salida en el rango [-1,+1].

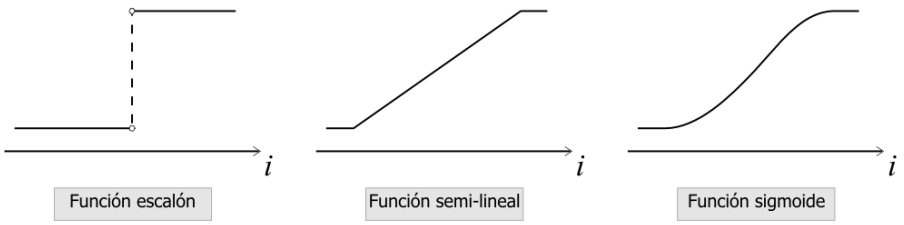


Figura 2.3: Ejemplo de funciones de activación para un nodo.

2.4.2. Topologías

Se destacan dos tipos de redes neuronales:

- **Redes progresivas** (o de alimentación hacia adelante): el flujo de datos es unidireccional desde la entrada hacia la salida. El procesamiento de estos datos se puede extender sobre múltiples (capas de) nodos, pero no existen ciclos.
- **Redes recurrentes** (o de alimentación hacia atrás): permiten conexiones bidireccionales y también ciclos.

Las redes más comunes están formadas por capas de unidades y generalmente son redes progresivas, donde las conexiones tienen un único sentido y no hay lazos. Además, las unidades de un nivel se comunican con todas las unidades del siguiente nivel, pero no existen conexiones entre unidades de un mismo nivel, ni con niveles anteriores, ni conexiones que salteen algún nivel [CFV04a].

El encontrar una estructura adecuada para la red es un problema en sí mismo, que puede ser visto como un problema de búsqueda. A veces suelen utilizarse otros mecanismos de aprendizaje (algoritmos genéticos por ejemplo) para resolver dicho problema y encontrar una estructura óptima para la red. Otras veces simplemente se parte de alguna red construida de antemano y se la va modificando hasta obtener una red adecuada al ambiente y sistema a utilizar.

2.4.3. Características generales de las redes neuronales

- Con una red neuronal se puede representar prácticamente cualquier función de un conjunto de atributos dados (entrada de la red). Algunas funciones simples pueden ser representadas con pocos nodos y sólo una capa oculta, mientras que otras funciones más complejas requieren de una red de varias capas ocultas.

- Pueden ser muy tolerantes al ruido en los datos de entrada, sin embargo suelen necesitar grandes cantidades de datos para poder generar una función que represente más o menos correctamente el ambiente del sistema.

- Las redes neuronales suelen funcionar como cajas negras, básicamente debido a que pueden poseer niveles ocultos de activación, y los pesos de conexión de los nodos son manejados internamente por la propia red según las funciones de ponderación. Una vez que la red se ha definido y entrenado, es capaz de predecir futuros comportamientos, sin embargo solo se puede conocer la salida a partir de determinada entrada, quedando oculto al resto del sistema lo que sucede dentro de la propia red.

2.5. Aprendizaje por Refuerzo

La noción de Aprendizaje por Refuerzo (*Reinforcement Learning - RL*) se conoce desde los primeros días de la cibernética y se utiliza tanto en estadística, psicología, neuro-ciencia y ciencia de la computación. Su promesa resulta interesante ya que es una vía para programar agentes a través de recompensas y castigos sin la necesidad de especificar cómo la tarea puede ser realizada [KLM96].

Aprendizaje por Refuerzo puede entenderse como el problema que resuelve un agente que debe aprender comportamiento a través de interacciones de ensayo y error en un entorno dinámico. Desde esta perspectiva resulta más apropiado pensar en *RL* como una clasificación de problemas que como un conjunto de técnicas.

Hay dos estrategias principales para solucionar problemas de *RL*. Una es buscar en el espacio de comportamientos hasta encontrar uno que resulte apropiado (dependiendo de lo que se espere como resultado) para el entorno en cuestión. Esta estrategia es utilizada en Algoritmos Genéticos y Programación Genética. Otra consiste en usar técnicas estadísticas y métodos de programación dinámica para estimar la utilidad de tomar ciertas acciones en determinados estados que presenta el entorno. Todavía no está claro cual de estas aproximaciones o estrategias es mejor y en que circunstancias [KLM96].

2.5.1. Aprendizaje y planificación

RL conjuga mecanismos de planificación con técnicas de aprendizaje. La planificación permite tomar una determinada decisión (acción), trazando un mapa de posibles situaciones futuras antes de que hayan sido verdaderamente experimentadas. Sin embargo, el efecto de las acciones no puede

ser totalmente predicho. Por lo tanto, el agente debe monitorear periódicamente su entorno para reaccionar adecuadamente.

2.5.2. Modelo de Aprendizaje por refuerzo

En el modelo estándar de RL, un agente está conectado al entorno por intermedio de sus percepciones y acciones. En cada interacción el agente recibe una entrada *(input)*, alguna indicación del estado actual del entorno, s; el agente elige una acción, a, y genera un *salida (output)*. La acción tomada modifica el estado del entorno de alguna forma, y el valor del cambio de estado es comunicado al agente a través de un valor escalar que se denomina señal de refuerzo (*reinforcement signal*), r. El comportamiento del agente debe ser tal que elija acciones que tiendan a maximizar la sumatoria de las señales de refuerzo recibidas a lo largo de toda la ejecución. Finalmente, el estado en el que el agente logra su objetivo o finaliza su ejecución se conoce como **estado terminal**.

Formalmente, el modelo consiste de:

- un conjunto discreto de estados del entorno, S
- un conjunto discreto de acciones del agente, A
- un conjunto de señales de refuerzo escalares; típicamente $\{0, 1\}$ o números reales

La figura 2.4 ilustra una simplificación del modelo estándar de aprendizaje por refuerzo presentado en [CFV04b].

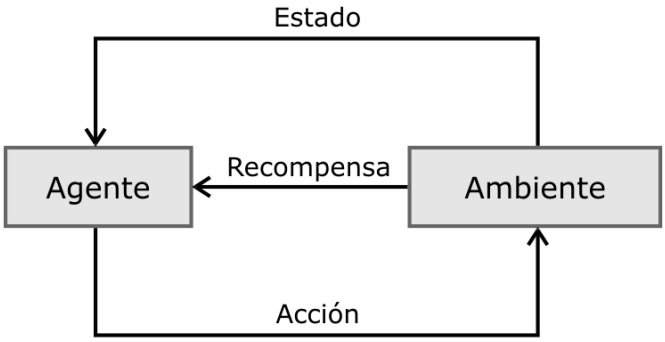


Figura 2.4: Modelo básico de Aprendizaje por Refuerzo.

La dificultad del aprendizaje de esta forma radica en que el agente no conoce que acciones pueden dar lugar a una recompensa positiva y cuales no, por lo que generalmente se trabaja bajo ensayo y error. Aquellas acciones que logren acercar al agente a su objetivo final serán valoradas positivamente y serán tenidas en cuenta en futuras acciones o tomas de decisiones. El agente debe ser lo suficientemente inteligente para poder proyectar a largo plazo sus objetivos, y evitar caer en la obtención de pequeños objetivos de corto plazo que pueden no beneficiarlo en el objetivo final o de largo plazo.

Dentro de RL tenemos dos tipos de agentes:

- **aprendiz pasivo:** se limita a observar el entorno y aprender de éste, almacenando la información en su base de conocimiento para utilizarla posteriormente.
- **aprendiz activo:** a medida que va aprendiendo del entorno que lo rodea, va tomando decisiones y realimentando su base de conocimiento con el resultado de dichas acciones.

2.5.3. Características de los algoritmos de Aprendizaje por Refuerzo

Todos los algoritmos de RL deben saber complementar la búsqueda de buenas acciones con la memoria de acciones pasadas. La experiencia (memoria) proporciona búsquedas de soluciones más eficientes. A su vez, la experiencia se nutre de la información obtenida de la búsqueda. Los agentes deben poder conectarse con el entorno, obtener información del mismo (a través de los sentidos, sensores) y tomar acciones que afecten su estado. Cada agente debe tener uno o varios objetivos, definidos en términos de cómo el entorno se comporta a lo largo del tiempo, bajo la influencia de sus acciones.

RL se diferencia de Aprendizaje Supervisado puesto que en el último los individuos aprenden bajo la tutela de un supervisor conocido (maestro) que explícitamente les dice qué deberían responder a las entradas. Algunos autores afirman que esta clase de aprendizaje -Aprendizaje Supervisado- no se adecua a las exigencias de un agente autónomo [SB04].

Cada agente debe encontrar un balance entre la explotación de su experiencia y la exploración de nuevas alternativas, que permitan mejorar las futuras tomas de decisiones.

2.5.4. Características de los agentes

En general, los agentes que implementan RL poseen cuatro componentes básicos [SB04] que se detallan a continuación y se muestran en la figura 2.5:

- política
- función de recompensa
- función valor
- modelo del entorno

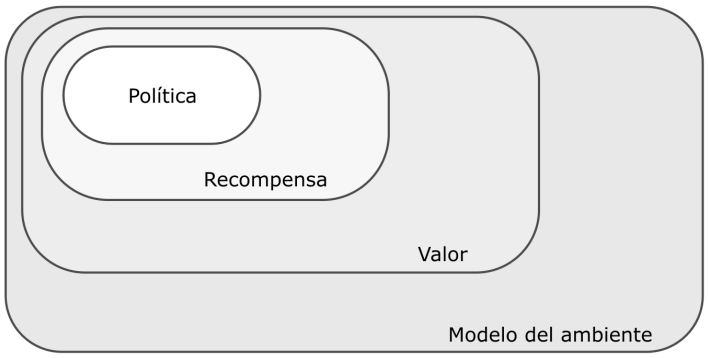


Figura 2.5: Componentes de un agente que implementa Aprendizaje por Refuerzo.

La política es la función de toma de decisiones de cada agente. Especifica qué acción debe tomarse en cada una de las situaciones. Es el núcleo del agente, ya que es ella quien determina totalmente su comportamiento. Los otros componentes sirven para cambiar y mejorar la política.

La función de recompensa define el objetivo del agente: maximizar la recompensa que recibe a lo largo de la ejecución. Las recompensas no son fijas; deben ser ajustadas acorde al problema al que se enfrenta el agente. Puede ser utilizada como la base para cambiar de política.

La función valor estima la recompensa: especificando una noción de qué es bueno en las futuras ejecuciones. Por ejemplo, bloquear al adversario que está intentando llegar a la pelota puede ser mejor que correr hacia ella. En este caso, la función valor estimó que moverse a una determinada posición (que bloquea al adversario) era mejor que competir por la pelota. Esto podría permitir que otro agente del equipo propio tengan más oportunidades de retomar el dominio del juego [CDB96].

La última característica es, como su nombre lo indica, un modelo del entorno en el que el agente se encuentra inmerso. Dada una situación y una acción, el modelo determina el próximo estado y las próximas recompensas posibles.

No todos los agentes utilizan un modelo del entorno. Aquellos métodos que no están basados en el entorno se denominan métodos libres del modelo. Los métodos libres del modelo son simples y pueden generalmente determinar comportamientos óptimos. Sin embargo, los métodos basados en el modelo encuentran soluciones más rápidamente (con menos experiencia) [SB04].

2.5.5. Modelos de comportamiento óptimo

Antes de comenzar a pensar sobre algoritmos que permitan aprender comportamientos óptimos, se debe decidir o definir un modelo de optimalidad. En particular, se debe especificar cómo el agente debe tomar en cuenta el futuro en las decisiones que realiza sobre cómo comportarse (que acciones tomar) en el presente.

Existen tres modelos que se destacan por concentrar la mayoría de los esfuerzos realizados en esta área.

Modelo de horizonte-finito (*Finite-horizon Model*)

Es el más fácil de pensar y consiste en que el agente en un momento dado debe optimizar las recompensas esperadas en los siguientes h pasos como se ilustra en la fórmula 2.5

$$E\left(\sum_{t=0}^{h} r_t\right) \tag{2.5}$$

sin preocuparse de lo que ocurra después.

En la ecuación 2.5, r_t representa la recompensa recibida t pasos hacia delante a partir de cero. Este modelo puede ser utilizado de dos formas, una donde la política de toma de decisiones siempre varía y otra donde el agente actúa siempre de acuerdo a la misma política. Sin embargo, el inconveniente con este modelo radica en que requiere que se tenga conocimiento previo sobre la duración del tiempo de vida del agente de antemano y esto no siempre es un dato conocido.

Modelo de horizonte-infinito devaluado (*Infinite-horizon discounted Model*)

Este modelo toma en cuenta todas las recompensas obtenidas por el agente a lo largo de su ejecución, pero las recompensas que son recibidas en los pasos futuros son devaluadas geométricamente de acuerdo a un factor de descuento γ, con $0 \leq \gamma < 1$, como se ilustra en la fórmula 2.6.

$$E\left(\sum_{t=0}^{\infty}\gamma^t r_t\right) \tag{2.6}$$

γ se puede interpretar de muchas formas. Puede ser visto como una tasa de interés, la probabilidad de seguir ejecutando durante otro paso, o simplemente un truco matemático para alcanzar la suma infinita.

Este modelo se considera matemáticamente más manejable que el Modelo de horizonte-finito, siendo esta una de las razones más importantes por las cuales este modelo ha recibido más atención [KLM96].

Modelo de promedio de recompensas (*Average-reward Model*)

En este modelo el agente elige las acciones que optimizan el valor promedio de recompensas tomando en cuenta toda la ejecución como se ilustra en la fórmula 2.7.

$$lim_{h\to\infty}E\left(\frac{1}{h}\sum_{t=0}^{h} r_t\right) \tag{2.7}$$

2.5.6. Medidas de desempeño del aprendizaje

Los criterios vistos en la sección 2.5.5 pueden ser utilizados para evaluar la calidad de las políticas aprendidas por el algoritmo de aprendizaje. También es útil disponer de criterios que permitan evaluar la calidad del aprendizaje propiamente.

Las siguientes son algunas medidas incompatibles que se utilizan con ese propósito.

Convergencia Eventual al Óptimo (*Eventual convergence to optimal*)

Un agente que alcanza un porcentaje de optimalidad de 99 % debe preferirse, en muchas aplicaciones, a uno que garantiza una optimalidad de 100 % eventualmente pero tiene la tasa de aprendizaje se alcanza muy lentamente.

Velocidad de Convergencia a la Optimalidad (*Speed of convergence to optimality*)

La optimalidad es un resultado considerado asintótico. Por lo tanto, Velocidad de Convergencia a la Optimalidad se aplica en la práctica a la velocidad de convergencia hacia una vecindad de la optimalidad. Por otro lado, este criterio introduce la necesidad de definir tanto una noción de distancia a la optimalidad, como el umbral a partir del cual se está suficientemente cerca de la misma.

Otra debilidad que tienen las estrategias que toman en cuenta este criterio es que cuanto más rápido se aproximan a la optimalidad, más errores pueden cometer durante el periodo de aprendizaje. Por otro lado, estrategias menos agresivas que tomen un poco más de tiempo en alcanzar la optimalidad, pueden lograr reforzar mejor el aprendizaje, y son preferibles [KLM96].

Arrepentimiento (*Regret*)

Una medida más apropiada, es considerar la disminución esperada en las recompensas obtenidas durante la ejecución del algoritmo de aprendizaje en relación con las que se obtendrían ejecutando un comportamiento óptimo desde el comienzo. Esta medida penaliza los errores donde sea que

ocurran durante la ejecución. Desafortunadamente, esta estrategia es más difícil de implementar que las anteriores.

2.5.7. Recompensas demoradas (*Delayed Reward*)

En general, en los problemas de *RL*, las acciones adoptadas por el agente determinan no sólo la recompensa inmediata sino el siguiente estado del entorno (al menos probabilísticamente). Por lo tanto, el agente toma en cuenta el siguiente estado y la recompensa cuando decide tomar una acción determinada. En estos casos el modelo de optimalidad considerado para toda la ejecución determinará cómo tomar en cuenta los valores obtenidos en el futuro. El agente debe ser capaz de aprender a partir de recompensas demoradas: o sea, puede obtener una secuencia de pequeñas recompensas inmediatas primero, para finalmente llegar a un estado donde obtener un valor de recompensa alto. Es decir, el agente debe poder aprender cual de las acciones son deseables tomando en cuenta la o las recompensas que pueden obtenerse en un futuro arbitrariamente lejano [KLM96].

Procesos de decisión Markovianos (*Markov Decision Processes*)

Los problemas con refuerzo demorado se pueden modelar adecuadamente como procesos de decisión de Markov (*MDPs*).

Un *MDP* consiste en:

- un conjunto de estados S
- un conjunto de acciones A
- una función de recompensa $\Re : SxA \rightarrow \Re$, y
- una función de transición de estados $T : SxA \rightarrow \Pi(S)$, donde un elemento de $\Pi(S)$ es una distribución de probabilidad sobre el conjunto S (es una correspondencia entre estados y probabilidades).

Según este modelo, *T(s, a, s')* es la probabilidad de ir desde el estado s hacia *s'* mediante la acción *a*.

La función de transición especifica probabilísticamente el siguiente estado del entorno en función del estado actual y la acción elegida por el agente.

La función de recompensa especifica la recompensa inmediata esperada en función del estado actual y la acción elegida por el agente.

El modelo es Marcoviano si las transiciones de estado son independientes de los estados anteriores del entorno y las acciones tomadas.

2.5.8. Aprendiendo políticas óptimas

En casos en los que el modelo es conocido se puede encontrar la política de decisiones óptima utilizando técnicas de programación dinámica. Sin embargo, *RL* se concentra primariamente en cómo obtener o alcanzar esa política óptima cuando el modelo es desconocido desde el comienzo.

Por lo tanto, el agente debe interactuar con el entorno directamente para obtener información que pueda ser procesada, cuya interpretación depende del algoritmo de aprendizaje utilizado, para producir una política óptima.

Existen dos formas de lograrlo:

- Libre de Modelo (*Model-free*): Aprender un controlador sin aprendizaje del modelo.

- Basado en Modelo (*Model-based*): Aprender un modelo y utilizarlo para derivar un controlador.

La respuesta sobre cual es mejor aún es objeto de debate en la comunidad de RL. En este documento se profundizará en Q-Learning, un algoritmo que utiliza el enfoque *Libre de Modelo*.

El mayor problema al que se enfrenta un agente que implementa RL es conocido como Asignación Temporal de Créditos (*Temporal Credit Assignment*). Se trata de responder a la siguiente interrogante: £Cómo saber si una acción que ha sido tomada es buena, considerando que pueda incidir en el logro del objetivo final?

Una estrategia posible podría ser esperar hasta el final de la ejecución para evaluar el resultado de haber tomado la acción y recompensarla si fue bueno o castigarla en caso contrario. Desafortunadamente, existen tareas para las cuales es difícil determinar el final, y se pueden requerir grandes cantidades de memoria. Por el contrario, se pueden usar la información de cada iteración para ajustar el valor estimado de un estado basándose en el valor de la recompensa inmediata y el valor estimado del estado siguiente. Los algoritmos que utilizan esta noción se conocen como Métodos de Diferencia Temporal (*Temporal Difference Methods*).

Q-Learning

Es un método de diferencia temporal que resulta particularmente atractivo de ser analizado ya que numerosos equipos de fútbol de robots han aplicado distintas variantes del mismo para atacar el problema de la adaptación de los agentes al entorno de juego [ATB04]. Aprender acerca de cómo opera el equipo contrario resulta esencial para que los agentes se comporten hábilmente. Bajo estas condiciones el aprendizaje por refuerzo puede aportar muchas ventajas respecto a otros tipos de aprendizaje.

A los efectos de comprender el algoritmo se definen algunas notaciones adicionales a las que se introdujeron en las secciones anteriores (2.5.2, 2.5.7).

Sea *Q*(s, a)* la recompensa devaluada por tomar la acción a estando en el estado s, y luego elegir las restantes acciones óptimamente. De modo que *Q*(s, a)* puede definirse recursivamente según la ecuación 2.8.

$$Q^*(s,a) = R(s,a) + \gamma \sum_{s' \in S} T(s,a,s') * \max_{a'} Q^*(s',a') \tag{2.8}$$

Se establece una función Q que asigna un valor a una pareja (acción, estado). Este valor indica qué tan beneficioso para el agente es emprender dicha acción en ese estado. Dicho valor se conoce como **valor Q** y se denota por *Q(a,s)*. A partir de estos valores se obtiene la función de utilidad como muestra la siguiente ecuación y se presenta en [CFV04a]:

$$U(s) = \max_{a} Q(s,a) \tag{2.9}$$

A partir de estos valores Q el agente puede aprender sin necesidad de contar con un modelo del entorno en el que se desempeña. La ecuación de restricción que utiliza el agente para actualizar sus valores Q es la siguiente [CFV04a]:

$$Q(s,a) = Q(s,a) + \alpha(R(s) + \max_{a'} Q(s',a') - Q(s,a)) \tag{2.10}$$

2.5.9. Aprendizaje por refuerzo en espacios continuos

Algunos problemas se desarrollan en ambientes dinámicos y con un espacio infinito de estados o acciones posibles. Muchos de los algoritmos de Aprendizaje por refuerzo asumen conjuntos discretos de estados y acciones, con lo cual no son aplicables a este tipo de entornos. A veces, las variables continuas son discretizadas para luego aplicar algún algoritmo de *RL*. Sin embargo, si se realiza una mala discretización de los estados o acciones, los resultados serán malos y la optimización de la política prácticamente imposible. Si se realiza una discretización muy fina, la cantidad de datos de entrenamiento necesaria crece considerablemente y se pierde generalización. En la práctica, el problema de aprendizaje en espacios continuos es atacado con técnicas de generalización que permiten almacenamiento compacto de la información aprendida y transferencia de conocimiento entre estados y acciones "análogos".

HEDGER

Esta sección presenta HEDGER [SK00], un algoritmo que aproxima la función de valor de *Q-learning* de forma segura para aprendizaje online de políticas de control dinámicas.

Aproximar la función de valor correctamente

En *Q-learning*, realizar la acción a en el estado s resulta en llegar al estado *s'* con un retorno r, y deriva en una actualización de la aproximación de la función valor según la ecuación antes vista 2.10.

El valor del par estado-acción (s,a) es actualizado de acuerdo al factor de aprendizaje α, el factor de descuento γ, y la aproximación actual del máximo valor esperado del siguiente estado *s'*. Este último generalmente es calculado como el máximo de los valores de las posibles acciones a partir de *s'*.

A partir de esto se observa que si se comete un error en la aproximación del valor de *s'*, el valor actualizado para la pareja (s,a) tendrá un pequeño error. Si luego continuamos usando este valor en sucesivos cálculos, el error se irá incrementando y nunca se obtendrá una buena aproximación.

Reducir el error de aproximación

Como varios estudios lo demuestran, una de las principales fuentes de error en el cálculo de la aproximación de la función valor es que ésta es muy propensa a la sobre-estimación. La mayor causa de esto parece ser lo que se conoce como *extrapolación oculta*. O sea, usualmente se suele interpolar los datos de entrenamiento pero no extrapolarlos. Se podrá decir si la función aprendida modela los datos de entrenamiento, pero no si los datos de entrenamiento en sí mismos, derivados del modelo, son correctos [SK00].

Una posible solución consiste en construir una especie de casco convexo alrededor de los datos de entrenamiento y solo responder consultas que caigan dentro de éste. El problema entonces se convierte en cómo definir dicho casco convexo. Cook sugiere que la mejor forma de resolverlo es a través de un casco hiper-elipsoide [SK00]. Para una matriz X, cuyas columnas corresponden a los puntos de los datos de entrenamiento, calculamos la *hat matrix* como se muestra en la ecuación 2.11.

$$V = X(X^{-1}X)^{-1}X^{-1} \tag{2.11}$$

Un punto arbitrario $\overline{x}$ pertenece al casco convexo si cumple la ecuación 2.12.

$$\overline{x}(X^{-1}X)^{-1}\overline{x} \leq \max_i v_{ii} \tag{2.12}$$

Donde v_{ii} es un elemento de la diagonal de V.

El Algoritmo predictivo HEDGER

En esta subsección se muestra el algoritmo presentado en [SK00].

HEDGER se basa en el principio de *Locally Weighted Regression* (*LWR*), una variación de la regresión lineal estándar, en la cual los puntos de entrenamiento cercanos al punto de consulta tienen mayor influencia sobre la superficie de la regresión que aquellos más lejanos. Una nueva regresión es calculada para cada punto consultado. Esto deriva en un modelo no lineal relativamente simple.

La función que asigna los pesos a los puntos de entrenamiento en base a la distancia a los puntos de consulta suele ser una función kernel, como por ejemplo una Gaussiana.

Una de las razones más importantes para elegir *LWR* en HEDGER es por lo simple y rápido de entrenar. Dado que *LWR* retiene todos los valores de entrenamiento, es posible reevaluar la aproximación realizada en base a los puntos de entrenamiento originales.

Para utilizar *LWR* para aproximar la función de valor es necesario combinar de alguna manera los vectores de estados y acciones. El resultado de esta combinación es usada como vector de entrada para el sistema *LWR*. Generalmente, para combinar los vectores de estados y acciones, se realiza la concatenación de ambos vectores.

La técnica de *LWR* estándar utiliza todos los datos de entrenamiento disponibles para hacer los cálculos, lo cual puede ser prohibitivo en algunos casos. Pero utilizando la técnica del casco convexo se puede mejorar considerablemente el desempeño. El algoritmo 1 muestra como funciona la predicción en HEDGER.

Algoritmo 1: Predicción de HEDGER

Entrada:
Punto para el cual se halla el valor Q: (s,a)
ancho de banda de LWR: h

Salida: Predicción del valor de la función: Q(s,a)

1. Concatenar s y a para formar $\vec{q}$
2. Encontrar conjunto de puntos K, más cercanos a $\vec{q}$ que k_{thresh}
3. if ($|k| \leq k_{min}$) then
4. retornar valor por defecto "don't know"
5. else
6. Calcular el IVH, H, basado en k
7. if ($\vec{q} \in H$) then
8. Calcular peso del núcleo, k_i para cada $k_i \in K$
9. $k_i = \exp(-(\vec{q} - \vec{k_i})^2/h^2)$
10. Realizar regresión sobre K usando los pesos k_i
11. retornar la función ajustada f(s,a)
12. else
13. retornar valor por defecto "don't know"

Se define un valor k_{thresh} para el *LWR* donde todo punto consultado que se encuentre a una distancia mayor que k_{thresh} de un punto de entrenamiento no será tenido en cuenta en la regresión. Para obtener una búsqueda eficiente de los puntos k que cumplen esta condición y se encuentran en el conjunto K se pueden utilizar las estructuras *kd-tree*. Generalmente se necesita que el conjunto K contenga tantos puntos como parámetros se deseen ajustar en el modelo de regresión.

En el algoritmo esto se hace utilizando el valor k_{min}. Luego se construye un *IVH* (casco convexo) alrededor de K y el punto consultado es comparado contra este casco. Si el punto esta dentro del casco, entonces se realiza el *LWR* normalmente, de lo contrario se retorna el *Q-value don't know*.

El algoritmo de entrenamiento de HEDGER

La experiencia sobre el mundo es modelada a través de una tupla de cuatro valores: *(s,a,r,s')*, donde a partir de la acción a se pasa del estado s al *s'* con un retorno *r*. El algoritmo 2 muestra como a partir de una tupla de experiencia del mundo se realiza la aproximación de la función valor.

Algoritmo 2: Entrenamiento de HEDGER

Entrada:

Experiencia: (s,a,r,s')
Valor de aprendizaje: α
Factor de descuento: γ
Ancho de banda de LWR: h

1. $q = Q_{predict}(s, a)$ usando el Algoritmo 1
2. $q_{next} = \text{máx}_{a'}\, Q_{predict}(s', a')$
3. K =conjunto utilizado en el cálculo de q
4. $k_i = \exp(-(\vec{q} - \vec{k_i})^2/h^2)$
5. $q_{new} = q + \alpha(r + \gamma q_{next} - q)$
6. LearnQ(s,a) = q_{new}
7. for each (s_i, a_i) en K do
8. $\quad Q(s_i, a_i) = Q(s_i, a_i) + k_i(q_{new} - Q(s_i, a_i))$

Primero se obtienen aproximaciones para el *Q-value* actual y para el máximo valor del estado destino *s'*.

En la linea 3 se guardan los puntos usados en la predicción *LWR* para usos futuros, y sus pesos asociados se cargan en la línea 4.

En la línea 5 se calcula la nueva aproximación de *Q(s,a)*, usando la regla de actualización de *Q-Learning* estándar. Este nuevo valor es usado entonces como un punto de entrenamiento supervisado en el subsistema *LWR*. El valor es "aprendido" simplemente con el hecho de almacenarlo en la memoria. Luego se actualiza cada uno de los puntos del conjunto K.

Esta forma de actualizar la aproximación permite mantener la función de valor suavizada (*smooth*).

En la línea 2 se busca el valor de la mejor acción del próximo estado *s'*. Sin embargo, como estamos en un espacio continuo, el estado nunca ha sido visitado. Por lo tanto, se debe buscar la mejor acción de una región de estados cercanos al estado *s'*.

El sistema comienza muestreando n acciones diferentes de los estados cercanos a *s'* y prediciendo sus *Q-values*. Entonces el algoritmo itera para ajustar la superficie cuadrática a los puntos muestreados y luego se obtiene un punto en el máximo de esta función ajustada. Cuando dos máximos sucesivos están más cerca que un valor límite dado, el algoritmo finaliza. Esta aproximación es sumamente dependiente de los puntos de entrenamiento iniciales y puede tener problemas con máximos locales en la función valor.

Suplantando el conocimiento inicial

Dado que inicialmente el sistema cuenta con pocos datos de entrenamiento, los agentes se comportarán aleatoriamente durante un tiempo hasta tener un "paso base" para la aproximación de la función valor.

Al comienzo se podrían cargar tuplas con valores obtenidos de pruebas reales ejecutados en el entorno real, o con porciones de código que controlen el sistema en esta etapa, o por manejo humano explícito de los agentes. En todos estos casos HEDGER comienza aprendiendo pasivamente a partir de la observación y guardando la información en la memoria. Luego de esta fase de entrenamiento, se supone que la función valor ha sido aproximada razonablemente y el sistema pasa a usar la política de control apropiada.

Capítulo 3

Sistemas Multi-agente

Dentro del paradigma de agentes, uno de los conceptos más atrayentes es la resolución de problemas mediante la utilización de Sistemas Multi-agente (*Multi Agent System - MAS*).

La mayor complejidad en los *MAS* se centra en la resolución de la comunicación y coordinación de los agentes que componen el sistema.

En la mayoría de las metodologías y estudios realizados sobre *MAS* se destacan dos conceptos importantes: modelo de roles y modelo de interacción. El modelo de roles define el comportamiento esperado de un agente del sistema en un momento dado, mientras que el modelo de interacción define como interactúan los agentes entre sí para lograr una buena cooperación y coordinación, necesarias para alcanzar los objetivos generales del sistema.

Los agentes pueden cumplir varios roles simultáneamente, ya sea porque es requerido por el protocolo de interacción en un mismo contexto o porque el agente esté participando de varios contextos en un momento dado.

La FIPA (*Foundation for Intelligent Physical Agents* - organización dedicada a generar estándares para sistemas de agentes), ha creado varios protocolos generales, que ha expresado en AUML para distintas interacciones recurrentes en el diseño de *MAS*, por ejemplo, las *subastas*. AUML es básicamente un dialecto de UML con algunos artefactos adicionales más apropiados al dominio de los *MAS*.

3.1. Elementos básicos de Sistemas Multi-agente

Existe una gran cantidad de ventajas para la elección de una arquitectura multi-agente que es inevitablemente distribuida debido a la autonomía característica de los agentes, aún no existiendo una distribución física de los mismos [CFV04a].

En [CFV04a] se plantean como posibles motivos para la elección de un *MAS* los enumerados en la siguiente lista:

- La facilidad de diseño o implementación.
- La distribución puede facilitar el descubrimiento de algoritmos que difícilmente tengan sentido en una arquitectura centralizada.
- La información y los sistemas se encuentran distribuidos y no pueden ser accedidos de forma centralizada (por problemas físicos, políticos, económicos, etc.)
- La complejidad de los sistemas participantes requiere un enfoque distribuido, por algunas de las siguientes razones:

    1. Los sistemas están geográficamente distribuidos

2. Los sistemas poseen muchos componentes
3. Los sistemas son fuertemente heterogéneos
4. Necesidad de soportar un contenido masivo, tanto en cantidad de conceptos, como en la información almacenada acerca de estos.
5. El dominio de la aplicación que debe ser cubierto por el sistema puede ser muy amplio.

Se pueden clasificar las diversas estrategias en cuatro grandes categorías: Modularidad, Distribución, Abstracción e Inteligencia [CFV04a]. Los sistemas de Inteligencia Artificial Distribuida (*Distributed Artificial Intelligence - DAI*) suelen utilizar un enfoque fuertemente modular (para facilitar la distribución del sistema) e inteligente, aplicando de forma simultánea las cuatro estrategias mencionadas.

La mayor parte de la investigación en Inteligencia Artificial, se ha centrado en la construcción de agentes que puedan actuar de forma inteligente, con únicamente un centro de control y razonamiento. Luego, al ver que los agentes no actúan solos y deben cooperar entre sí, se abre la puerta al estudio de sociedades de agentes inteligentes.

3.1.1. Mensajes y Comunicación

Dado que el sistema se compone de más de un agente, es sumamente necesario establecer un sistema de comunicación entre los mismos. Esta comunicación permite además la cooperación de todos los agentes para lograr los objetivos globales del sistema, más allá de que cada agente posea sus propios objetivos individuales, los cuales pueden ser conocidos por los demás agentes del sistema o no.

En [CFV04a] se presentan tres conceptos importantes vinculados con la comunicación entre agentes en un *MAS*:

- La **coordinación** es una propiedad de un sistema de agentes desarrollando una actividad en un entorno compartido. El grado de coordinación que exhibe un sistema está dado por cuanto los agentes evitan tener un comportamiento extraño ante sus pares, reduciendo conflictos por acceso a recursos, evitando *deadlocks* y *livelocks* (inanición).
- La **cooperación** es la coordinación entre agentes no antagónicos, mientras que la negociación es lo propio entre agentes competitivos o simplemente interesados en sus propios objetivos (*self-interested*).
- La **coherencia** es una medida de qué tan bien un sistema actúa como una unidad. Uno de los problemas a resolver al construir un *MAS*, es mantener una coherencia global, prescindiendo a su vez de un control global explícito.

Siempre que sea posible, un *MAS*, y sus participantes, deberían escoger mensajes convencionales, especialmente teniendo en cuenta que típicamente estos sistemas son abiertos, en donde otros agentes pueden ser introducidos en cualquier momento.

Los mensajes no siempre pueden ser comprendidos por sí solos, sino que deben ser interpretados teniendo en cuenta los estados mentales de los agentes, el estado del ambiente, y la historia de cómo éste llegó al estado presente. Las interpretaciones están directamente influenciadas por los mensajes anteriores y las acciones de los agentes.

Cuando la comunicación ocurre entre varios agentes, su significado es dependiente de las identidades y roles de los participantes, y cómo estos son especificados o mencionados. Adicionalmente, un mensaje puede ser enviado a un agente o a un conjunto que satisfaga determinada cantidad.

Para que la comunicación inter-agente sea de utilidad, éstos deben ser capaces de establecer diálogos, o secuencias de mensaje. En un dialogo, un agente puede asumir un rol activo, pasivo o ambos, permitiéndoles actuar en forma de maestro (*master*), esclavo (*slave*) o colega (*peer*) [CFV04a].

3.1.2. Protocolos de comunicación entre agentes

Los protocolos de comunicación son los responsables de gobernar las secuencias de mensajes que son intercambiados por los agentes, esto es, las conversaciones que ellos mantienen. Como ya fue mencionado, los objetivos de los agentes y su propensión a coordinar, competir o ignorar a los otros agentes de su entorno, condicionan los protocolos de interacción que utilizarán.

En [CFV04a] se presentan una serie de protocolos que los agentes deben cumplir para asegurar una correcta comunicación entre agentes de un mismo sistema:

- Protocolos de coordinación
- Protocolos de cooperación
- Negociación

A continuación se resumen brevemente algunos puntos importantes dentro de cada uno de los ítems mencionados anteriormente.

3.1.3. Protocolos de Coordinación

La distribución del control implica que los agentes tienen cierto grado de autonomía para generar las acciones que crean convenientes y decidir qué objetivos perseguirán en un momento dado.

Uno de los principales usos de la coordinación de agentes es el de resolver de forma conjunta una serie de objetivos, usualmente llamados **Objetivos del sistema**. Las actividades que los agentes deben realizar en el proceso de alcanzar objetivos conjuntos, puede ser modelada como la búsqueda en un grafo de objetivos *AND/OR*. Este grafo incluye las dependencias de los objetivos y recursos que son necesarios para cubrir los objetivos primitivos, los que corresponden a las hojas del grafo.

Al modelar de esta forma los *MAS*, se logra que las actividades que requieren coordinación queden perfectamente identificadas. Algunas de estas tareas se listan a continuación:

- Definir el grafo de objetivos, actividad que incluye la identificación y clasificación de dependencias.
- Asignación de regiones del grafo a los agentes apropiados.
- Controlar las decisiones que afectan cuáles áreas deben ser exploradas.
- Navegar a través del grafo.
- Asegurarse que un recorrido exitoso del grafo es reportado a todos los agentes.

El determinar cuales de estas tareas deben ser realizadas por uno o más agentes, es una cuestión de diseño del *MAS*.

3.1.4. Protocolos de Cooperación

Una estrategia básica compartida por muchos protocolos de coordinación es la de descomponer el problema en tareas más pequeñas que luego son distribuidas a los agentes miembros del sistema.

Algunos criterios para la asignación de las tareas, una vez descompuestas, pueden verse a continuación:

- Evitar la sobrecarga de recursos críticos.
- Asignar tareas a agentes con capacidades apropiadas.

- Hacer que un agente que posea una visión global, realice la distribución.
- Asignar responsabilidades solapadas a algunos agentes para lograr coherencia global.
- Asignar tareas altamente interdependientes a agentes semántica o físicamente contiguos para minimizar los costos de comunicación y sincronización.
- Reasignar tareas para completar objetivos urgentes.

Redes de Contratos

Las **redes de contratos** (*contract net)*, son uno de los protocolos de cooperación más utilizados para la resolución de problemas de forma cooperativa. En su estructura básica, se considera a un agente que requiere un servicio de otros, como el **administrador** (*manager*) y aquellos que puedan ser capaces de resolver estas tareas, son llamados potenciales **contratistas** (*contractors*).

Desde el punto de vista de un administrador, el proceso que el protocolo establece es el que sigue:

- Anunciar una tarea que debe ser realizada.
- Recibir y evaluar ofertas de los potenciales contratistas.
- Asignar el contrato a el o los contratistas escogidos.
- Recibir y sintetizar los resultados.

Por su parte, los contratistas ven el proceso de la siguiente forma:

- Recibir anuncios de tareas.
- Evaluar la capacidad para realizarlas.
- Responder (ofertar o declinar).
- Realizar la tarea si la oferta es aceptada.
- Reportar los resultados.

Los puntos claves en el diseño de un protocolo particular que siga este enfoque son por ejemplo:

- Establecer las estrategias de publicación y re-publicación de tareas por parte de los administradores.
- Definir si una negociación se llevará a cabo o no en el momento de la asignación de un contrato para definir características específicas.
- Afinar los tiempos de validez de ofertas y publicaciones.
- Seleccionar los criterios utilizados por los agentes para la evaluación de las ofertas y publicaciones de tareas.

Pizarrón

El protocolo de resolución cooperativa de problemas llamado pizarrón (*blackboard*), está basado en la siguiente metáfora:

"*Un conjunto de expertos en diferentes áreas del conocimiento que, a partir de un problema planteado en un pizarrón, monitorean a este último buscando una oportunidad para aportar su experiencia en el desarrollo de una solución. A su vez la solución es publicada en el mismo pizarrón. De esta forma, los aportes de los participantes son tomados como datos por los demás en una dialéctica que eventualmente lleva a una solución consensuada*".

3.1.5. Negociación

Una definición usual de negociación presentada en [CFV04a] dice: "Una negociación es el proceso por el cual dos o más agentes llegan a un acuerdo, mientras cada uno trata de alcanzar un objetivo particular."

En un *MAS* podría suceder que dos o más agentes encuentren conflictos a la hora de desarrollar sus tareas para lograr sus objetivos personales. En este caso los agentes deberán negociar para que ambos puedan alcanzar sus objetivos particulares y, además, para poder llegar al objetivo general del sistema si es que lo hay. En estos procesos de negociación es donde se deben utilizar los distintos protocolos de coordinación y cooperación para que éstos interactúen y puedan llegar a un acuerdo común.

Cuando la negociación o coordinación debe realizarse entre muchos agentes, puede ser sumamente ineficiente la interacción de todos con todos, por lo que pueden utilizarse mecanismos de mercado para la interacción. Por ejemplo podrían implementarse sistemas económicos para la asignación de recursos a los agentes. Todo recurso que un agente pueda necesitar u ofrecer se le asigna un precio. Los agentes que necesitan recursos consultan el mercado para conocer los precios y agentes que ofrecen dicho recurso y así interactuar directamente con éstos para obtener lo que necesita. Pueden utilizarse teorías económicas y de mercado para modelar este tipo de comunicaciones.

Capítulo 4

Algoritmos evolutivos

En esta sección se describirán algunos argumentos y/o fundamentos que hacen interesante y válida la aplicación de algoritmos evolutivos para lograr buenos resultados a nivel de la estrategia de colaboración y cooperación de los equipos de fútbol de robots, evolucionando los componentes de software donde se realiza la toma de decisiones dentro de los mismos.

También se mencionarán algunos ejemplos exitosos que demuestran la validez de los argumentos que se irán vertiendo. Asimismo, se relacionará Programación Genética (*Genetic Programming - GP*) con otras áreas de investigación donde, partiendo de otros enunciados, se busca alcanzar el mismo objetivo.

4.1. Programación Genética en fútbol de robots

El contexto de la aplicación (torneos de fútbol de robots) es un entorno muy dinámico, donde los datos de entrada pueden contener imprecisiones y donde la toma de decisiones se ve fuertemente condicionada por las restricciones de tiempo. Cuando se habla del contexto simulado, en particular las interferencias en la información o la fidelidad de la misma, puede transformarse en un factor realmente importante a tener en cuenta. Surgen otras limitaciones como a nivel de las comunicaciones entre robots y la cantidad y tipo de información recibida que puede estar limitada dependiendo del simulador.

Básicamente, la idea es que para tener éxito los jugadores de cada equipo deben cooperar para superar a sus adversarios.

Uno de los mayores problemas que se interponen a la hora de utilizar este tipo de técnica de aprendizaje es que requieren un gran número de evaluaciones y cada evaluación supone una nueva simulación. Para fijar ideas, en el caso de *GP* aplicada al fútbol de robots, podrían requerirse un número de evaluaciones del orden de las 100.000 para obtener una solución razonable. Obviamente, existen y se han probado, con suficiente éxito, formas de reducir el tiempo necesario desde 1 año (evaluaciones de 5 minutos) hasta semanas o meses [Luk98].

Un aspecto importante cuando se pretende hacer evolucionar el comportamiento de un equipo de fútbol es contar con un conjunto de funciones de comportamiento (acciones elementales) de bajo nivel para que los jugadores puedan utilizarlas. En este punto vale la pena detenerse ya que las características de estas funciones deben ajustarse al problema que se pretende resolver; tampoco deben presentar problemas o trabas al proceso evolutivo. Por lo tanto, definir con qué y con cuáles funciones se cuenta, y algunas de sus características propias, lejos está de ser una tarea menor, sino que puede comprometer toda posibilidad de evolución. Una de las características que pueden generar inconvenientes es la generalidad, vale decir, no pueden ser demasiado genéricas ya que eso podría enlentecer extremadamente la convergencia del proceso o incluso hacerla inalcanzable. Por otro lado, ésta es una característica deseable dado que funciones menos específicas pueden aportar

comportamientos más ricos o de mayor complejidad. Lo que habitualmente resulta satisfactorio es una combinación de funciones específicas y genéricas en alguna medida intermedia.

4.1.1. Criterios de evolución

Una vez que se cuenta con las funciones básicas, es necesario adoptar un criterio para aplicar *GP* a los efectos de evolucionar el comportamiento del equipo. Una forma puede ser construir el equipo a partir de los individuos que se logre evolucionar. O sea, se hace evolucionar un programa (jugador) y luego se arma el equipo con la cantidad de jugadores (todos iguales) que se necesite. Otra alternativa es hacer evolucionar un solo genoma que representa todo el equipo.

Estos dos enfoques ponen de manifiesto que se pueden lograr dos tipos de resultados: equipos homogéneos y heterogéneos desde el punto de vista del comportamiento aprendido por sus integrantes (jugadores). Con el enfoque homogéneo cada jugador ejecutará exactamente el mismo algoritmo que sus compañeros de equipo. Con una aproximación heterogénea, cada jugador puede desarrollar y seguir luego un algoritmo único y propio.

Las diferencias entre estos enfoques se resumen a continuación:

1. El tiempo requerido por un enfoque heterogéneo es mayor ya que en el caso homogéneo solamente se hace evolucionar un genoma, mientras que el heterogéneo necesita evolucionar tantos genomas como individuos haya en el equipo.

2. En un dominio donde la heterogeneidad pueda ser una propiedad deseable, la aproximación heterogénea puede contribuir a una mayor flexibilidad y promete comportamientos y cooperación especializada.

En [Luk98] se desarrollaron aproximaciones híbridas que intentan acceder a la heterogeneidad sin invertir tanto tiempo en evaluaciones costosas. La idea es evolucionar subgrupos dentro del equipo siguiendo un enfoque heterogéneo intra-grupos y un enfoque homogéneo inter-grupos. Un ejemplo de esto se puede observar en la figura 4.1.

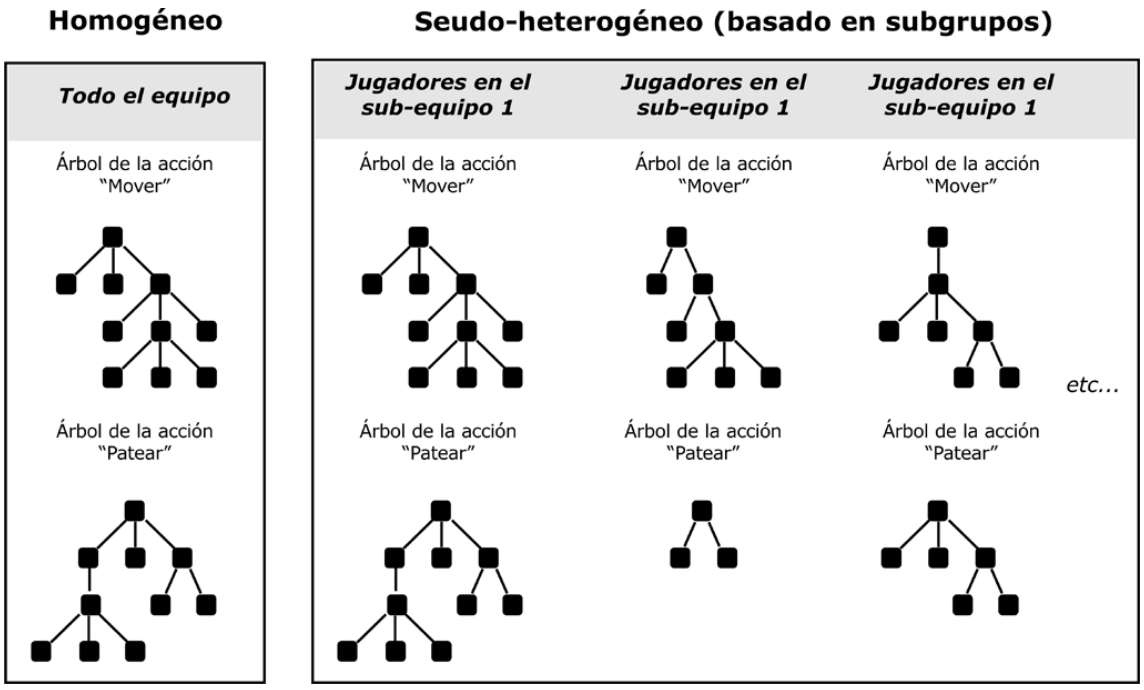


Figura 4.1: Codificación de los genomas de un equipo de fútbol de robots.

4.1.2. Función de fitness

Otro aspecto importante a considerar cuando se utiliza *GP* es determinar cómo evaluar la calidad (*fitness*) de las soluciones intermedias que se van generando (poblaciones). Se requiere una función, conocida como función de *fitness* que, tomando en cuenta determinados aspectos de las poblaciones, pueda indicar si se evoluciona y en qué medida.

En principio, surge una opción bastante intuitiva y es hacer competir las soluciones intermedias contra equipos hechos a mano sobre los cuales se conozca, por ejemplo, el nivel de dificultad que presentan determinados aspectos del juego. Dichos equipos pueden ser existentes o creados específicamente para las pruebas. El problema con esta opción es que, por un lado, se conoce empíricamente que las estrategias evolutivas logran mejores resultados cuando la dificultad del problema crece a la vez que el proceso de evolución avanza. O sea, a medida que la población mejora, el problema se vuelve más difícil y esto es poco probable de lograr con adversarios codificados a mano. Por otro lado, al enfrentar siempre a los mismos oponentes se corre el riesgo de evolucionar hacia estrategias específicas para derrotar a cierta clase de equipos, perdiendo el resultado toda generalidad.

Otra posibilidad es desarrollar competencias entre las poblaciones (*Competitive Fitness Enviroment*) para evaluar los genomas. Esta opción puede ser costosa en tiempo de evaluaciones dependiendo del tipo de competencia elegida. Por ejemplo, una competencia donde juegan todos contra todos requeriría un número igual a $\frac{n^2-n}{2}$ evaluaciones. Si se juegan dos partidos entre pares de una misma población el número de evaluaciones sería igual a $(n-1)$. Si se juega un solo partido se evalúan un total de $\frac{n}{2}$ partidos.

A pesar de conllevar un costo más elevado, esta alternativa cumple con que la dificultad del problema al que se enfrentan las poblaciones va en ascenso ya que los integrantes de una población que va evolucionando ofrecen un nivel de dificultad similar. Puede promover la generalidad ya que

cada equipo está obligado a superar adversarios diferentes cada vez.

Las funciones de *fitness* son caóticas y ocasionalmente pueden tener efectos indeseables sobre las poblaciones de soluciones. Por ello es muy importante elegir una función que permita al proceso converger hacia una solución sin permanecer oscilante ni en forma prematura arrojando resultados sub-óptimos.

Algunas experiencias [Luk98] demuestran que funciones de *fitness* que contemplan muchos aspectos del juego (demasiados parámetros) provocan una convergencia prematura arrojando como resultado estrategias de juego demasiado pobres. Por otro lado, si solo se concentra en aspectos centrales como puede ser el tanteador o la diferencia de goles entre equipos, se logran resultados muy buenos. Un resultado empírico relacionado con esto es que a pesar de que a primera vista una función con estas características podría parecer excesivamente simple o burda (por lo tanto generaría un problema en caso que inicialmente se obtuvieran demasiados empates), las pruebas indican que los primeros equipos tienden a ser muy ofensivos y pobres en defensa dando como resultados tanteadores muy altos. Esta situación tiende a revertirse con el transcurso del proceso evolutivo de aprendizaje, haciéndose los equipos casi tan fuertes en defensa como en ataque.

4.1.3. Trabajos conjuntos

Como se mencionó al comienzo de esta sección también es interesante mostrar cómo se relacionan estas técnicas de aprendizaje con otras que contribuyen al logro de equipos de fútbol de robots con comportamientos complejos y alto grado de cooperación entre sus integrantes. En tal sentido, se las relacionará con los conceptos del capítulo 3.

La intención es entonces, describir cómo se puede utilizar conjuntamente *GP* con *MAS*, adaptando un enfoque clásico de *GP* a un problema de aprendizaje con *MAS*, que pueda descomponerse jerárquicamente.

Quizás el reto más grande es encontrar la forma de la función de *fitness* y cómo aplicarla en el procesos evolutivo de un *MAS*.

Una aproximación a este logro se presenta en [HG02] y consiste en realizar una evaluación en capas o jerárquica donde se evalúa inicialmente a un nivel intermedio (pares de agentes o subgrupos) y luego se toman las poblaciones resultado como semillas para evolucionar el equipo entero.

Aprendizaje en Capas (*Layered Learning*) es un término utilizado en la literatura de Aprendizaje Automático y Agentes Inteligentes para describir una forma o línea de desarrollar tareas, a veces de forma incremental, para adquirir o lograr comportamientos jerárquicos basados en aprendizaje por refuerzo (ver sección 2.5).

Aplicar esta línea de trabajo para resolver un problema supone partir el problema en una jerarquía de sub-problemas, resolviendo el problema original a partir de los resultados aprendidos secuencialmente en las capas inferiores y utilizados como entrada por las superiores.

Esta aproximación de aprendizaje en capas muestra una propiedad particularmente atractiva y es que posibilita y facilita el descubrimiento de comportamientos primitivos en pequeños grupos que pueden ser reutilizados y/o adaptados para lograr metas más complejas compartidas por todo el equipo.

Otro beneficio que se ha comprobado empíricamente es la mejora significativa en la velocidad de aprendizaje.

Estos enfoques tienen parte de su origen en observar el hecho de que el fútbol de robot es un *MAS* que está basado en un juego real de humanos y, por lo tanto, es importante tomar en cuenta la forma en que estos últimos desarrollan sus estrategias de juego. Precisamente, una observación importante es notar que un equipo se estructura sobre una jerarquía de comportamientos individuales, de pares y de pequeños grupos de jugadores.

A diferencia de otras aproximaciones más tradicionales de *GP* que describen un camino para la implementación de estructuras en la representación de los agentes, ésta describe la forma de entrenar un agente inteligente, brindando además, un camino para construir soluciones usando

una estrategia *Divide&Conquer*. Asimismo, la estructura de la solución alcanzada difiere de otras logradas con *Divide&Conquer* en que la solución final no tiene necesariamente por qué reflejar el procedimiento jerárquico de entrenamiento.

El método de descomposición utilizado es *bottom-up* y la justificación se sostiene en que el aprendizaje, básicamente el comportamiento cooperativo de *MAS*, a menudo ocurre en ese mismo sentido (*bottom-up*) ya que individuos o pequeños grupos de agentes primero aprenden a realizar tareas menores o menos complejas para luego aprender a realizar tareas de mayor complejidad y más sofisticadas.

Capítulo 5

Sociedades de agentes

5.1. Organización en Sistemas Multi-agentes

Con el afán de lograr cohesión en el comportamiento global que apunte a un objetivo común, los MAS tienden a ser concebidos dentro de una organización. Este concepto tiende a ser más importante en sistemas de agentes autónomos, ya que la autonomía individual sin organización colectiva puede transformarse en un obstáculo para la obtención de los objetivos del sistema. Por lo tanto, una de las motivaciones para introducir el concepto de organización en un MAS es limitar la autonomía de los agentes [Hüb05].

En virtud de que la organización va de la mano con el concepto de rol, otras motivaciones son, facilitar el proceso de toma de decisiones (asumir un rol limita el espacio de búsqueda) y facilitar el razonamiento y la interacción con otros agentes (asumir un rol es información útil para los demás).

La organización compromete a los agentes. Estos compromisos pueden ser externos -o sociales- (con el resto de la organización) o internos (individuales). Entonces, los compromisos sociales pueden verse como formas flexibles de restringir el comportamiento de agentes autónomos.

5.1.1. Tipos de organización

Existen dos formas de ver la organización de un sistema [Hüb05].

- **Subjetiva:** Se basa en la observación. De esta forma cada observador puede interpretar a su manera la organización según lo que percibe.
- **Institucionalizada:** Existe una definición formal de la organización.

A partir de estas dos formas se puede clasificar las organizaciones en cuatro tipos como se muestra en la figura 5.1.

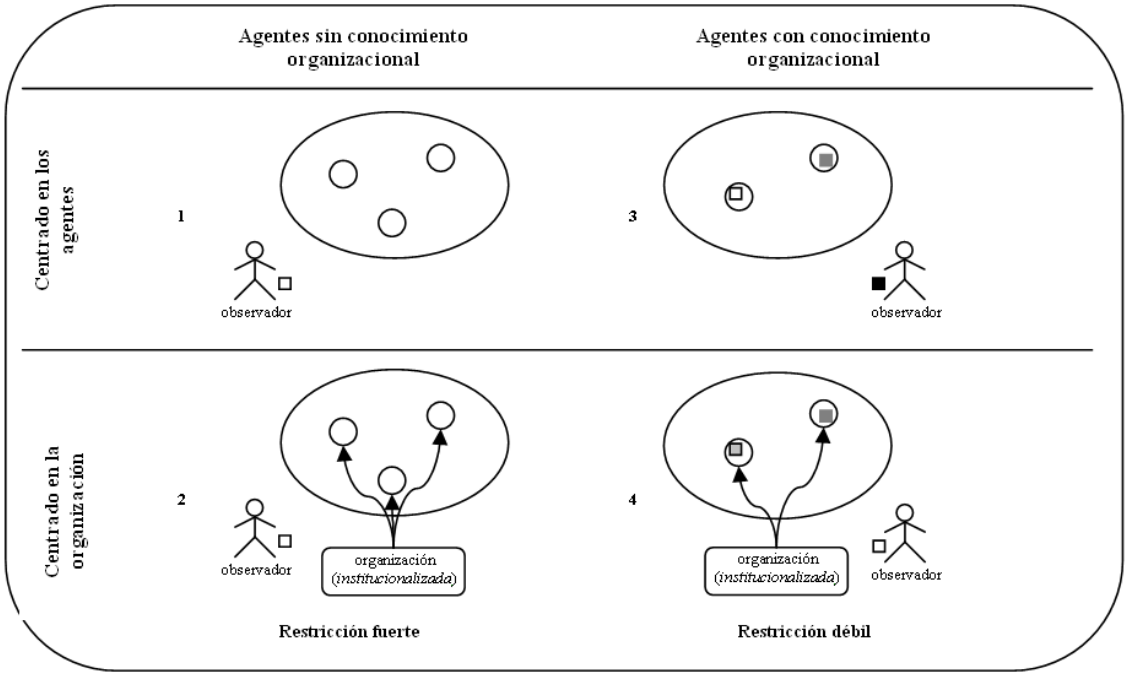


Figura 5.1: Tipos de organización.

1. Los agentes no conocen la organización pero un observador externo puede reconocerla (Ej. un hormiguero).

2. Restricción fuerte. Los agentes siguen la organización a pesar de no ser conscientes de su existencia (Ej. el agente está programado para seguir la organización pero no reflexiona sobre eso).

3. El agente observa e infiere la organización pero ésta no es explícita (Cada agente interpreta la organización de acuerdo a sus propios criterios, lo que puede generar múltiples interpretaciones).

4. Restricción débil. La organización existe explicitamente y los agentes pueden razonar sobre ella. (Pueden obedecer o revelarse).

5.2. Inspiraciones Biológicas

Las conclusiones y estudios realizados sobre sociedades de agentes se basan en observaciones realizadas al comportamiento cooperativo en comunidades de animales como hormigas, abejas y pájaros.

En el área del fútbol de robots incluso se ha llegado a estudiar la compatibilidad con sociedades de depredadores, asociando determinadas conductas al instinto competitivo de los robots.

También se ha comenzado a estudiar el comportamiento imitativo de animales más grandes para aprender nuevas habilidades, así como el comportamiento de los insectos respecto a la interconectividad física que permite una navegación colectiva sobre terrenos grandes y complejos.

5.2.1. Sociedades de hormigas

Las sociedades de insectos, y en particular las colonias de hormigas, son sistemas distribuidos que, además de las particularidades de sus individuos, presentan una organización social muy estructurada. Como resultado de esta organización, las colonias de hormigas pueden realizar tareas que en muchos casos exceden las capacidades de un único individuo. [DS04]

5.2.1.0.1. Comunicación entre hormigas Las hormigas utilizan básciamente sustancias químicas para comunicarse entre sí. Para ello, hacen uso del ambiente, en este caso el suelo, como intermediario en la comunicación. Por ejemplo, cuando una hormiga recolectora encuentra una fuente de alimento, retorna al nido dejando un rastro de feromona[1] en su camino. Esta feromona atrae a otras hormigas, las que buscarán la fuente de alimento y regresarán al nido dejando también su propio rastro químico sobre el mismo camino. De esta forma, la concentración de feromona es reforzada por cada hormiga, atrayendo cada vez a más individuos. La feromona, al ser un componente químico, posee la particularidad de evaporarse con el tiempo. Esta característica deterina que los caminos marcados por una hormiga, que no fueron seguidos por otras hormigas o por ella misma, desaparecerán con el tiempo. Así, los caminos óptimos o más cercanos al alimento se irán reforzando y remarcando hasta tanto no se agote la fuente de alimento, mientras que los caminos más largos o menos recorridos se desvanecerán lentamente.

5.2.1.0.2. Modelos y algoritmos basados en hormigas El área de "algoritmos de hormigas" o *ant algorithms* estudia modelos derivados de la observación del comportamiento real de las hormigas, y usa estos modelos como fuente de inspiración para el diseño de nuevos algoritmos para la resolución de problemas de optimización y control distribuido. La idea principal es utilizar los principios de organización que permiten un comportamiento bien coordinado en las sociedades reales de hormigas, para potenciar la coordinación de agentes artificiales que colaboran para resolver problemas computacionales. Muchos de los aspectos del comportamiento de las colonias de hormigas han inspirado diferentes tipos de algoritmos. Algunos ejemplos son: recolección de alimento, división de tareas, *brood sorting*, y transporte cooperativo. En todos estos ejemplos, las hormigas coordinan sus actividades mediante *stigmergy* [2], una forma indirecta de comunicación que se realiza mediante la modificación del ambiente. Los biólogos han demostrado que muchos de los comportamientos a nivel de colonia, observados en las sociedades de insectos, pueden ser explicados por simples modelos en los cuales solo está presente la comunicación *stigmergy*. En otras palabras, los biólogos han demostrado que a menudo es suficiente considerar la comunicación indirecta para explicar como se organizan las sociedades de insectos. Basado en esto, la idea de los *ant algorithms* es utilizar una forma de *stigmergy* artificial para coordinar sociedades de agentes artificiales. Uno de los ejemplos más exitosos de *ant algorithms* es el *Ant Colony Optimization* (ACO)[3][DS04].

[1] Una feromona, es alguna sustancia o conjunto de sustancias químicas producidas por algunos organismos vivos que las utilizan para transmitir mensajes a otros miembros de la misma especie. Hay varios tipos de feromona, como por ejemplo feromonas de alarma, feromonas de rastros al alimento, feromonas sexuales, y muchas otras que afectan el comportamiento de los individuos de la especie. Estas sustancias son muy utilizadas por los insectos para comunicarse, colaborar y organizarse. También algunas especies de vertebrados y plantas utilizan feromonas para comunicarse, pero no han sido tan estudiados como lo insectos.

[2] El término *stigmergy* fue presentado por Grassé para describir una forma de comunicación indirecta realizada a través de modificaciones en el ambiente que observó en la casta de trabajadores de dos especies de termitas. [DS04]

[3] El algoritmo *Ant Colony Optimization* fue desarrollado por Moyson y Manderick en 1988 y luego continuado por Marco Dorigo. Consiste en una técnica probabilística para la resolución de problemas computacionales que se reducen a encontrar un camino óptimo en un grafo. La idea detras de este algoritmo es simular el comportamiento de las hormigas para encontrar un camino óptimo entre el nido y la fuente de alimento, llevado a un sistema computacional, en el que hormigas artificales se mueven en un grafo, buscando el camino óptimo para ir de un nodo a otro. Los algoritmos de *Ant Colony Optimization* han sido usados para encotnrar soluciones sub-óptimas a problemas de transporte urbano o ruteo de paquetes en redes, y presenta la ventaja de poder ejecutar continuamente y adaptarse a cambios en tiempo de ejecución. [DS04]

5.2.2. Decisiones Colectivas

En los sistemas conformados por varios robots que cooperan para lograr objetivos comunes, a menudo se hace referencia al comportamiento colectivo o global del sistema y también al comportamiento individual de los robots que lo componen.

El comportamiento del sistema es la acumulación de los comportamientos individuales de cada robot. En este contexto, decir que un sistema colectivo de robots (*Collective Robotic System*) ha tomado una decisión, implica que cada individuo que conforma el sistema deberá actuar o comportarse de forma tal de ser consistente con la decisión tomada.

Este aspecto cobra especial relevancia cuando se refiere a sistemas con control descentralizado de acciones. Es decir, cada integrante del colectivo tiene la posibilidad de resolver una situación determinada y necesita comunicar su solución a los demás integrantes para que ésta sea adoptada por el todo (sistema). De modo que el desafío es poder lograr consensos en la toma de decisiones y que las mismas se tomen con cierto grado de confiabilidad.

Los equipos de robots que se utilizan en juegos, como el fútbol, son ejemplos claros de sistemas colectivos que en algunos casos no tienen una única fuente de control. En este contexto, para tener éxito, la toma de decisiones del equipo debe ser rápida y coherente.

Para aproximarse a una solución a este problema de toma de decisiones colectivas, algunos investigadores tomaron como referencia el comportamiento de algunas especies de hormigas. En particular, la especie *Leptothorax albipennis* sirvió de inspiración por el comportamiento que presenta la colonia de hormigas cada vez que necesitan cambiar de hormiguero [PZ04]. En el caso de esta especie se comprobó que cada cierto tiempo el hormiguero se deteriora de tal forma que se hace necesario que la colonia entera sea reubicada.

Las colonias de la *Leptothorax albipennis* por lo regular están compuestas por 300 especímenes, aproximadamente. Al momento de tomar la decisión de mudarse a otro hormiguero debe lograrse el consenso para evitar que la colonia se parta en subgrupos.

Cuando el hormiguero se daña, algunos de los habitantes comienzan a buscar un nuevo lugar para construir el nuevo hormiguero. Cuando una hormiga encuentra un lugar potencialmente bueno debe evaluar su calidad y después retornar al hormiguero para que otras hormigas puedan inspeccionarlo. En los hechos, se lleva a cabo una especie de reclutamiento por parte de la hormiga que haya encontrado un lugar factible. Si ésta logra convencer a un número tal de hormigas de que este lugar es el mejor candidato a ser el nuevo hormiguero, habrá un consenso y luego se implementará el cambio transmitiéndole a las restantes la decisión tomada.

Un aspecto importante en este comportamiento es establecer el quórum mínimo (*quorum threshold*) necesario para poder adoptar una decisión. Otro aspecto importante es el hecho de que los lugares de mayor calidad serán bien considerados por las otras hormigas y, por lo tanto, la probabilidad de lograr un consenso es mayor cuanto mejor sea el sitio visitado. Como contrapartida, desde el punto de vista de la calidad, los sitios más pobres no convencerán a muchas hormigas dejándose de lado con el transcurso del tiempo.

Este comportamiento sirvió de inspiración para el algoritmo presentado en la figura 5.2 para toma de decisiones en sistemas colectivos de robots.

Este algoritmo muestra cómo, a través del reclutamiento sucesivo, las mejores soluciones ganan adherentes más rápido que las peores. Los robots con mejores soluciones son mejor dotados para seguir reclutando adeptos, mientras que los que proponen soluciones pobres tienden a quedar ociosos. Cuando se alcanza un número mínimo de robots que están de acuerdo, la solución es elegida por el sistema como un todo y se implementa.

Este algoritmo de toma de decisiones se parece a otros algoritmos como *Ant Colony Optimization* (mencionado anteriormente), pero existe una diferencia importante entre ellos. En *ACO*, el entorno se ve modificado porque se depositan rastros de feromona mientras las hormigas (*ant*

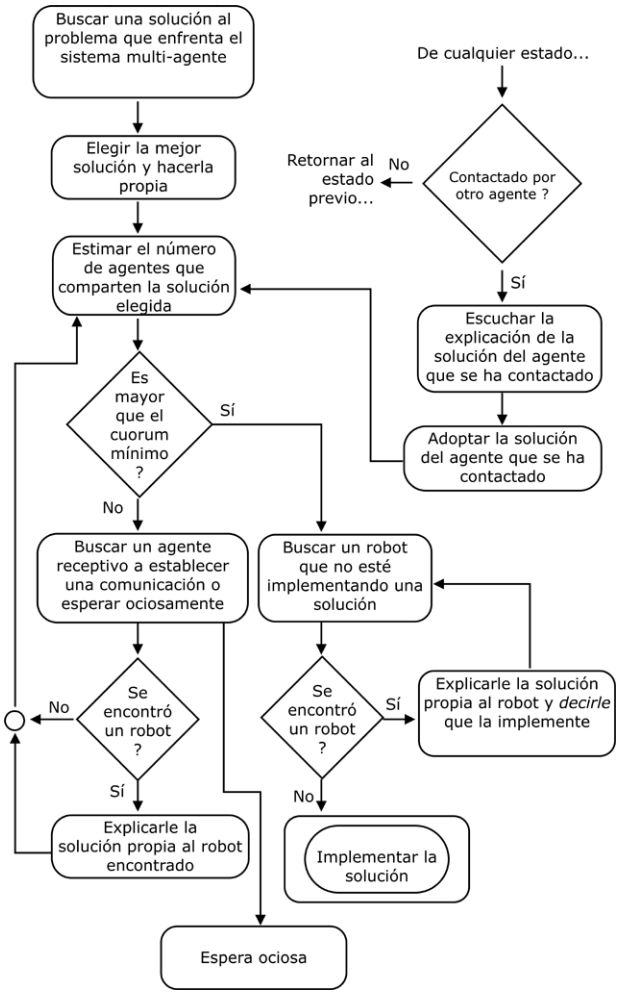


Figura 5.2: Flujo que ilustra un algoritmo distribuido de toma de decisiones (*Distributed Making Decisions*)

s*oftware*) recorren el espacio de soluciones posibles. Este rastro es utilizado para marcar el camino a otras hormigas hacia soluciones intermedias. En este caso, la probabilidad de que el rastro se intensifique crece para las mejores soluciones. En este algoritmo, en cambio, no se modifica el entorno. No hay ningún tipo de feromona o análogo. Los robots reclutan a otros directamente y así incrementan la popularidad de una solución. De esta forma se implementa un mecanismo más directo prescindiendo de nociones químicas que lo hacen impráctico en el caso de los robots reales [PZ04].

Capítulo 6

Reconocimiento de patrones

6.1. Introducción

El ser humano es capaz de percibir datos sensoriales a través de sus sentidos y reconocer objetos, conceptos, características, comportamientos, fenómenos, etc. En otras palabras, el procesamiento e interpretación de los datos sensoriales pretende obtener una descripción concisa y representativa del universo observado[1].

Los elementos del universo se perciben como patrones y los procesos que llevan a su comprensión son llamados *procesos perceptuales*. El etiquetado de esos elementos es lo que se conoce como *reconocimiento de patrones*. Por lo tanto, el reconocimiento de patrones es una herramienta esencial para la interpretación automática de datos sensoriales. Por ejemplo, el sistema nervioso humano recibe aproximadamente 10^9 bits de datos sensoriales por segundo y la mayoría de esta información es adquirida y procesada por el sistema visual. El procesamiento de imágenes de escenas complejas es un proceso en múltiples niveles que comprende dos tipos de metodologías:

- Reconocimiento de patrones basado en atributos.
- Reconocimiento de patrones basado en la estructura.

6.2. Modelo

En lo que refiere al modelo del sistema de reconocimiento de patrones del ser humano, se encuentra que los procesos perceptuales pueden ser modelados como un sistema de tres estados:

- adquisición de datos sensoriales
- extracción de características
- toma de decisiones

En [dTdI06] se propone dividir el problema del reconocimiento automático de una manera similar al reconocimiento realizado por los seres humanos y como lo muestra la figura 6.1.

- **Sensor:** Proporciona una representación factible de los elementos del universo a ser clasiifi-cados.

[1] Por observable se entiende todo aquello que sea perseptible. No se restringe únicamente a los objetos que pueden ser vistos mediante el sentido de la vista.

Figura 6.1: Etapas de un sistema de reconocimiento de patrones.

- **Extracción de Características:** Se realiza la eliminación de información redundante e irrelevante y se obtiene la información discriminatoria a partir del patrón de representación.
- **Clasicador:** Comprende la toma de decisiones del sistema. Su rol es asignar la categoría apropiada a los patrones de clase desconocida.

Según [PP02], el reconocimiento automático de patrones puede modelarse como una tarea de dos etapas consistentes en: aprender las propiedades comunes e invariantes de una conjunto de muestras que caracterizan una clase, y decidir que una nueva muestra es un miembro de una clase en base a las propiedades en común con las demás muestras del conjunto. Por consiguiente, la tarea del reconocimiento de patrones por medio de una computadora puede ser descrita como una transformación de un espacio de muestras M al conjunto de características F y finalmente al espacio de decisiones D.

6.3. Diseño

En lo que refiere al diseño de sistemas de reconocimiento de patrones, se presentan algunos problemas que alejan la posibilidad de encontrar una solución óptima. Teóricamente, se podría diseñar un sistema de reconocimiento de patrones óptimo, por ejemplo, mediante la aplicación directa de la teoría de Bayes[2], si se conociera en su totalidad las características estadísticas del modelo en el proceso de generación de los patrones. Sin embargo en la práctica ésto no es posible. Los problemas identificados en[dTdI06] son los siguientes:

- el modelo no se puede conocer totalmente y/o
- la complejidad del sistema a diseñar está restringida por consideraciones económicas (*hardware*, tiempo).

6.4. Aprendizaje

En general, la base de conocimiento disponible para el diseño de un sistema de estas características es un conjunto de entrenamiento[3] constituido por observaciones, ya sean etiquetadas o no.

Si las observaciones se encuentran etiquetadas, se asume que para cada patrón o vector de observaciones en el conjunto de entrenamiento, un experto asigna una etiqueta con la clase correcta. El diseño de un sistema basado en un conjunto de datos clasificados de antemano se conoce como *aprendizaje supervisado*.

Si no se dispone de conocimiento experto sobre el conjunto de datos, o si el etiquetado de los patrones de entrenamiento es impracticable, entonces el problema de diseño implica la necesidad de una primera etapa de análisis de los datos. Este proceso primario de análisis se conoce como *aprendizaje no supervisado*.

[2] El teorema de Bayes relaciona las probabilidades condicional y marginal de eventos estocásticos
[3] Conjunto de datos reales que se utilizan para entrenar y ajustar el sistema.

En general, dado un conjunto de entrenamiento, el diseño del sistema de reconocimiento de patrones implica [dTdI06]:

1. Inferencia del modelo a partir de los datos (aprendizaje).
2. Desarrollo de reglas de decisión prácticas.
3. Simulación y evaluación del rendimiento del sistema.

6.5. Reconocimiento de patrones en el fútbol de robots

La predicción del comportamiento futuro de un oponente en entornos competitivos suele ser dificultosa. Una de estas dificultades radica en que generalmente los planes del adversario se encuentran ocultos y no se desea que sean descubiertos. El fútbol de robots es un ejemplo de entorno competitivo, altamente dinámico, multi-agente y de tiempo real, donde las decisiones deben ser tomadas rápidamente y de forma lo más precisa posible. Esta precisión en la toma de decisiones puede ser potenciada con la habilidad de predecir el comportamiento del oponente.

6.5.1. Aplicación en RoboCup

En [BW03] se propone un enfoque en capas para la resolución del problema de la predicción del comportamiento oponente en RoboCup:

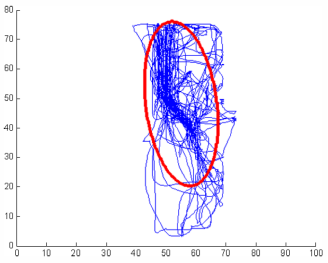
- **clasificación**: razonar acerca del comportamiento actual del oponente
- **modelado**: razonar acerca del proceso interno de decisiones.
- **precicción**: razonar acerca del comportamiento futuro.

En base a este enfoque se plantean las siguientes interrogante £Qué tan bien puede clasificar, modelar y predecir el comportamiento oponente un método probabilístico, teniendo en cuenta que el entorno es competitivo, multi-agente, altamente dinámico y parcialmente observable? y £qué efecto tienen estas predicciones sobre el desempeño del sistema de planificación? En [BW03] se proponen las respuestas mediante la construcción de un equipo para la categoría F180 de RoboCup, cuyo dominio se caracteriza por ser parcialmente observable, estocástico, secuencial, dinámico, contínuo y multi-agente, considerado como uno de los dominios con mayor dificultad para un sistema inteligente según Russell y Norving [RN02].

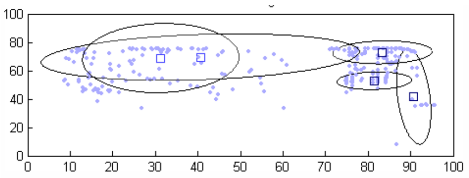
En este equipo, la clasificación del comportamiento oponente se realiza por medio de un *Clasificador Bayesiano* [4], donde las clases representan el comportamiento y los atributos representan el estado del juego. Este clasificador ofrece un rango de probabilidades de los comportamientos típicos en el ambiente. Por otra parte, el comportamiento oponente es modelado mediante una distribución probabilística 2D construida incrementalemnte en base a las características de los comportamientos, representando la función de utilidad de los oponentes. Finalmente, la predicción del comportamiento oponente combina el comportamiento actual, obtenido de la clasificación, con los modelos de comportamiento.

El reconocimiento se realiza por cada robot individualmente. No se analiza el comportamiento grupal del equipo o la interacción entre los comportamientos individuales.

[4] Un Clasificador Bayesiano es un método probabilístico de clasificación, construido sobre una red bayesiana capaz de clasificar clases de datos. [DH73]



a) ejemplo de estimación de la zona de movimiento de un robot.



b) ejemplo de *clusterización* de configuraciones (sólo se toman las posiciones de los robots).

Figura 6.2: Ejemplos de datos relacionados al comportamiento de un equipo (extraido de [LF04]).

6.5.2. Aplicación en FIRA

En [LF04] se presenta una aplicación de reconocimiento de patrones en el fútbol de robots. Más concretamente, se propone una solución al problema de identificar el comportamiento oponente para la categoría *SimuroSot* de la FIRA.

La predicción del comportamiento se realiza en dos instancias. Una relacionada con la estimación del área de movimiento de los distintos robots en una ventana de tiempo, clasificando los robots según su modo de juego (defensa, atacante, izquierdo, derecho, etc.). La otra consiste en extraer comportamientos grupales del equipo, determinando las formaciones más comunes, en base a sus posiciones tomadas como puntos de un espacio multidimensional y agrupando las formaciones similares, y con mayor tendencia a aparecer, mediante el algoritmo FCM - *Fuzzy C-Means Clustering* [LF04]. A partir de esta información se analizan las secuencias de eventos, generando grafos probabilísticos o cadenas de Markov que describen el comportamiento del equipo, especialmente en jugadas de peligro.

Para determinar el área de movimiento de los robots, la aplicación utiliza un modelo probabilístico basado en la distribución normal para estimar el área de influencia de cada uno, tomando como hipótesis que existen jugadores que siguen un movimiento aproximadamente rectilíneo. Con este modelo planteado se han obtenido buenos resultados para aquellos robots que se mueven en zonas estrechas. Sin embargo, en otros casos, se han observado movimientos elípticos, concluyendo que la distribución normal no ofrece buenos resultados en dichos casos.

En lo que refiere a la determinación de la formación del equipo, se define que una configuración de posiciones puede ser vista como un punto en un espacio de 32 dimensiones (posiciones (x,y) de los 10 robots, rotación (z) de los 10 robots y posición (x,y) de la pelota).

La figura 6.2 muestra un ejemplo de detección de movimiento y un ejemplo de *clusterización*.

Luego de obtenido un conjunto de configuraciones, pueden detectarse similitudes entre éstas utilizando un algoritmo de agrupamiento o *clusterización*. Una vez generados los clusters se construye un grafo de transiciones que modela de cierta forma las secuencias de configuraciones detectadas durante el juego. Cada nodo del grafo corresponde a un *cluster*. Dos *clusters* están relacionados si el partido tiene dos configuraciones inmediatas que "caen" dentro de cada uno de estos *cluster*.

Capítulo 7

Casos de Estudio

7.1. FRUTo

7.1.1. Introducción

El equipo FRUTo fue desarrollado en la Facultad de Ingeniería de la Universidad de la República Oriental del Uruguay. Como un proyecto de grado formó parte de los primeros pasos dados en el área de la robótica en Uruguay. Junto a este proyecto se desarrollaron otros dos que incursionaron en el área de Reconocimiento y Procesamiento de Imágenes y en el área de Construcción: Visión de Robots y Construcción de Robots de Bajo Presupuesto, respectivamente.

FRUTo se dedicó exclusivamente al comportamiento, desarrollando algoritmos para el ambiente simulado que proporciona la FIRA.

Por estos motivos, el equipo FRUTo resulta una referencia ineludible ya que fue la primera experiencia de desarrollo de comportamiento para robots realizada por la Facultad de Ingeniería y con la que se tiene mayor contacto.

Algunos de los conceptos, fórmulas e imágenes que se verterán en las secciones siguientes fueron extraídos directamente de la documentación producida por los integrantes del proyecto que desarrolló el equipo FRUTo [CC04].

A continuación se describen las características más sobresalientes de la implementación del equipo FRUTo. Estas características son las más destacadas si se pretende tomar al equipo FRUTo como referencia directa para realizar el comportamiento de un equipo de fútbol de robots.

7.1.2. Arquitectura

El estilo de arquitectura es muy similar al estilo en capas. Se compone de una jerarquía de módulos o componentes de *software* que determinan el comportamiento del equipo (comportamiento reactivo), y componentes separados de la jerarquía (por esta razón no es fiel al estilo en capas) que aportan consideraciones de carácter más deliberativo o de planificación.

Con esta disposición de los componentes de software se busca aprovechar las características de un estilo en capas, adaptada al fútbol de robots, para lograr comportamientos reactivos y contar con elementos deliberativos que permitan introducir cierta cuota de planificación. Los elementos deliberativos que se mencionan son: registro histórico o *tracking* (registro de situaciones o estados de juego útiles para ayudar en la toma de decisiones) y búsquedas en pequeños espacios de dominio de planificación.

Otro aspecto importante relacionado con el estilo de arquitectura elegido es que se adopta un enfoque vertical de una pasada. Es decir, toda la jerarquía se considera como un elemento que toma la información del mundo real desde un único punto de entrada, procesa la información y

emite una salida. Esta decisión se fundamenta en tratar de obtener un mejor desempeño y que la fuente de información es única.

La jerarquía propuesta consiste en seis niveles de abstracción que se describen a continuación comenzando por la capa de más bajo nivel.

Capa 0

Tiene la responsabilidad de enviar la información de salida a los robots. La salida se compone de las velocidades derecha e izquierda de las ruedas de cada robot, siendo estas los únicos datos de salida del sistema.

Capa 1

Encapsula los algoritmos de control de movimientos (*Path planning*). Estos algoritmos tienen la responsabilidad de determinar los comandos necesarios para ir de un punto a otro a través de cierta trayectoria llegando con cierta orientación.

Capa 1,5

Tiene la responsabilidad de implementar el control de colisiones. Las colisiones no son tenidas en cuenta en la planificación realizada en la capa 1 y por lo tanto esta capa debe detectar e intentar corregir este tipo de situaciones.

Capa 2

Encapsula un conjunto de habilidades básicas que son útiles (sirven de insumo) a las capas superiores para implementar el comportamiento y la colaboración entre los robots.

Son comportamientos de bajo nivel que se pueden combinar para lograr cooperación. Como ejemplo se citan: patear al arco, marcar, realizar un pase, moverse a una zona libre de la cancha, etc.

Capa 3

Define el concepto de rol como un conjunto de habilidades o comportamientos ofrecidos por la capa 2 tomando la idea de rol del fútbol de humanos. Intenta reproducir la noción de jugadores con habilidades específicas según el lugar de la cancha donde desarrollan el juego la mayor parte del tiempo. Ejemplos de roles serían: Defensa, Mediocampista, Atacante.

Capa 4

Tiene la responsabilidad de definir la estrategia del equipo. Por estrategia en este caso se considera la asignación de roles a cada robot y la política de intercambio dinámico de roles.

Además de estas seis capas que conforman la jerarquía de capas, se describen a continuación los elementos que se utilizan para introducir planificación o deliberación al comportamiento del equipo.

Módulo de Tracking

Tiene la responsabilidad de registrar la información histórica de todos los elementos que se definan como relevantes para el juego. Un ejemplo de ello son: posiciones, velocidades, comportamientos activados.

Observación: podría utilizarse para realizar estimaciones sobre la estrategia del rival siempre que se considere una ventana de tiempo pequeña.

Módulo de predicción

Tiene la responsabilidad de estimar estados futuros de los robots propios tomando en cuenta el estado actual en cada paso y una secuencia de comandos.

Las estimaciones no tienen una validez muy larga en el tiempo a causa del dinamismo propio del juego.

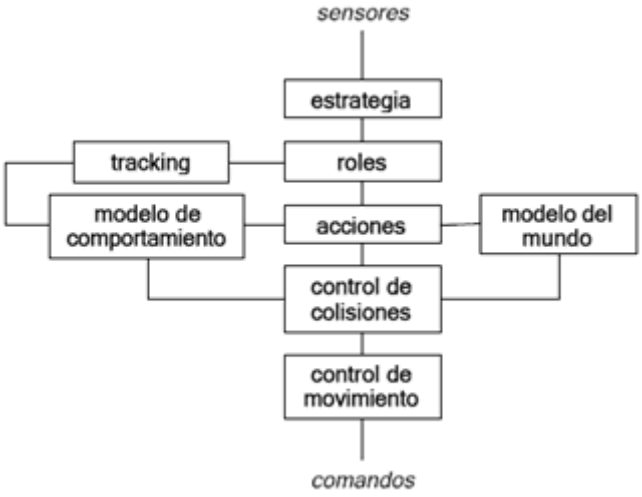


Figura 7.1: Arquitectura del equipo FRUTo.

La arquitectura mostrada en la Figura 7.1 representa la arquitectura de cada uno de los robots del equipo FRUTo. En este esquema global cada robot toma su decisión de juego teniendo en cuenta los datos de entrada y el comportamiento que asumirán sus compañeros de equipo.

Todo se fundamenta en la idea de que en cada paso cada robot sabe de forma aproximada la estrategia de juego que asumirá el equipo en el siguiente paso y por lo tanto toma sus decisiones según lo que entiende puede hacer mejor y lo que pueden hacer mejor los restantes robots.

La idea es que exista un razonamiento colectivo homogéneo que haga posible cierta cooperación y colaboración entre los integrantes del equipo para lograr alcanzar los objetivos del grupo.

7.1.3. Planificación de movimientos

La decisión de qué tarea debe ejecutar cada robot se determina en una máquina de estados especificada por el proyectista. Dependiendo del estado del juego, se fija el robot y el conjunto de tareas pasibles de ser ejecutadas. Para cada tarea del conjunto, se realiza una predicción para determinar cuál será la más exitosa a partir del momento actual y se escoje para ser asignada al robot.

El resultado de este proceso, la asignación de una acción a cada robot, es enviado al conjunto de funcionalidades encargadas de realizar la planificación de movimientos. Su responsabilidad es, en última instancia, determinar las velocidades para las ruedas izquierda y derecha que permitan cumplir con la tarea de manera eficaz y eficiente. La figura 7.2 muestra el flujo de ejecución del mecanismo de toma de decisiones y el algoritmo de planificación de movimientos.

A continuación se describen los conceptos involucrados en la planificación de movimientos.

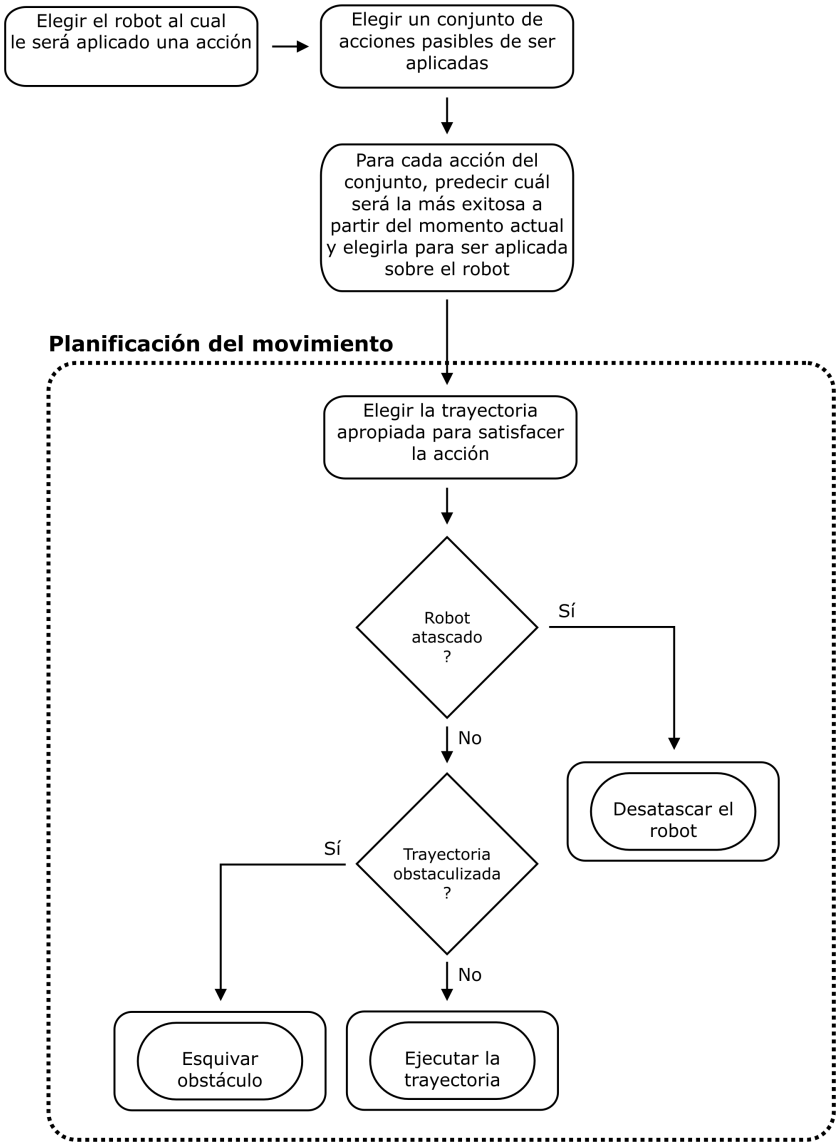


Figura 7.2: Diagrama de flujo del mecanismo de toma de decisiones y algoritmo de planificación de movimientos del FRUTo.

7.1.3.1. Acciones y controles

Se definen dos términos relacionados con la planificación de movimientos: *acciones* y *controles*. Una *acción* es el comportamiento que adopta un robot ante una determinada configuración del entorno, determinando el movimiento que debe realizar. Un *control* es un algoritmo que, a partir de la configuración actual del robot, genera la trayectoria necesaria para moverlo hacia una configuración tomada como destino.

Acciones

Las acciones se pueden clasificar en simples, combinables y coordinadas[1]. Las primeras son autónomas: no tienen en cuenta otros robots para ser ejecutadas satisfactoriamente. Ejemplos de éstas son: *Shoot* -tiro al arco-, *Intercept Back* -interceptar la pelota en la última línea defensiva- o *Intercept Goalie* -interceptar la pelota en la línea del arco-. Las segundas también son autónomas, pero se diferencian de las primeras en que intencionalmente intentan generar situaciones de juego que puedan ser aprovechadas por otras acciones. Ejemplos de éstas son: *Shoot Death* -tiro "de la muerte" al arco, que es aprovechada por *Relocate Shoot* -reposicionarse luego del "tiro de la muerte"-; o *Clear* -despejar la pelota-, que es aprovechada por *Relocate Clear* -reposicionarse para quedar cerca de la pelota luego de un despeje-. Las terceras necesitan de la coordinación de dos robots para que su ejecución sea satisfactoria. Ejemplos de éstas son: *Pass* -pasar la pelota hacia otro jugador-, que coordina con *Receive* -recibir la pelota de otro jugador-.

La elección del robot al cual se le aplicará una acción queda en manos del proyectista: es él quien fija el robot y el conjunto de acciones pasibles de ser aplicadas. Luego, cada acción posee un mecanismo interno de evaluación que permite predecir el grado de éxito que logrará el movimiento. La acción que evalúe más alto será la candidata final a ser aplicada. Si bien las acciones se definen como una habilidad que se realiza en el corto plazo, el valor resultado de la función de éxito debe mantener su validez durante un plazo suficiente que permita al robot tener continuidad en sus movimientos. Estas asignaciones de acciones a robots se realizan a nivel de la estrategia y fuera de los límites de la planificación del movimiento (ver figura 7.2).

Las acciones que cada robot del equipo FRUTo puede realizar están definidas en el cuadro 7.1.

[1] Esta clasificación fue definida a partir del análisis de la solución del proyecto FRUTo con fines aclaratorios.

Cuadro 7.1: Cuadro de Acciones definidas para el equipo FRUTo.

Acción	Definición
Shoot	Tirar al arco. Patear la pelota al arco.
ShootDeath	El "Centro de la muerte". Esta acción es una jugada preestablecida que intenta imitar a la jugada del mismo nombre en el fútbol tradicional. Dicha jugada consiste en el envío de un centro rastrero hacia el área en donde estará esperando precisamente el ejecutante de esta acción. Éste, una vez en condiciones de hacerlo, pateará la pelota hacia el arco con el objetivo de convertir el gol.
ShootPenal	Ejecutar un tiro penal.
Pass	Pasar la pelota. Patear la pelota para que un compañero la reciba.
Receive	Recibir un pase.
Clear	Despejar la pelota. Patear la pelota hacia una zona libre.
ClearH	Despejar la pelota horizontalmente, es decir con ángulo 0.
InterceptBack	Interceptar la pelota en la última línea defensiva colocándose entre la pelota y el arco propio.
InterceptFront	Interceptar la pelota en la primer línea defensiva colocándose entre la pelota y el arco propio.
InterceptGoalie	Atajar. Interceptar la pelota tomando como referencia el arco propio. Alternativa 1.
DefendGoal	Atajar. Interceptar la pelota tomando como referencia el arco propio. Alternativa 2.
RelocateShoot	Reposicionarse para esperar el rebote luego de un tiro al arco.
RelocateClear	Reposicionarse de manera tal de quedar cerca de la pelota luego de un despeje (Clear).
RelocateClearH	Análogo a RelocateClear pero para la acción ClearH.
RelocateDefendBack	Acción de defensa por defecto a ser ejecutada por un delantero (posición de atrás).
RelocateDefendFront	Acción de defensa por defecto a ser ejecutada por un delantero (posición de adelante).
RelocateBack	Acción de defensa por defecto a ser ejecutada por un defensa (posición de atrás).
RelocateFront	Acción de defensa por defecto a ser ejecutada por un defensa (posición de adelante).

Una vez asignada, la acción determina qué control o controles aplicar para generar la trayectoria adecuada para lograr el objetivo.

Controles

A partir de la configuración actual del robot, el control calcula las próximas velocidades de las ruedas para permitirle avanzar un paso hacia el objetivo. Para realizar este cálculo, el control se vale únicamente de la configuración actual del robot y de la configuración objetivo a la cual se quiere llegar, entendiéndose por configuración la posición y la rotación del robot. La sucesión de todos los pasos, desde que comenzó hasta que terminó la aplicación del control, determina la trayectoria del robot.

Se definen varios controles, donde cada uno tiene su propia forma de resolver la trayectoria hacia el objetivo. A continuación se enumeran y describen los controles definidos, mientras que en

[CC04] se detalla el funcionamiento de cada uno.

- *Control LV*

  Este control fue adaptado para el FRUTo y básicamente consta de dos fórmulas (ver ecuación 7.1) que determinan la velocidad lineal y angular del robot para llegar a un destino determinado.

$$\begin{aligned} &\nu = \gamma e \\ &\omega = \gamma \left(\sin\alpha + \tfrac{h\theta \sin\alpha}{\alpha} + \beta\alpha\right) \\ &\gamma > 0,\ 1 < h,\ 2 < \beta < 1 + h \end{aligned} \tag{7.1}$$

  ν =Velocidad lineal

  ω =Velocidad angular

  e =Distancia Euclidiana al punto destino.

  θ =Ángulo entre el eje horizontal y la recta que une el centro del robot y el punto destino.

  α =Diferencia entre θ y la orientación del robot.

  Valores utilizados de los parámetros:

  $\gamma = 2{,}0$

  $h = 2{,}0$

  $\beta = 2{,}5$

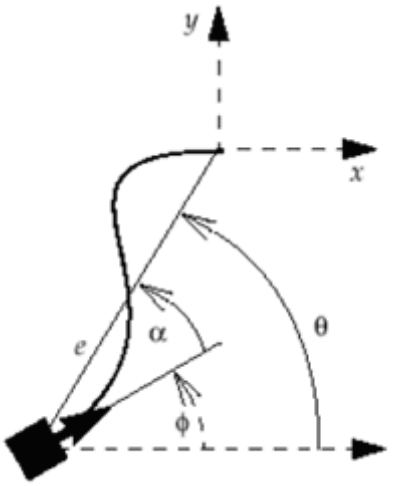


Figura 7.3: Parámetros - Path Planning

Este método asume que el objetivo (x_g, y_g, f_g) es (0, 0, 0) con f_g el ángulo de llegada, y que la posición actual del robot se encuentra en el cuadrante II o III de la cancha.

Por lo tanto, las velocidades izquierda y derecha (de cada rueda) del robot se determinan como se muestra en la ecuación 7.2.

$$\begin{array}{l} \upsilon_r = \nu + \frac{\omega L}{2} \\ \upsilon_l = \nu - \frac{\omega L}{2} \end{array} \tag{7.2}$$

Con L la distancia entre las ruedas del robot.

La adaptación que se le hace al control LV original tiene que ver con el hecho de que éste considera la velocidad proporcional a la distancia. Esto puede no ser aplicable en todos los casos ya que frente a distancias muy grandes la velocidad calculada superaría el límite (velocidad máxima) del robot, y frente a distancias muy pequeñas, la velocidad calculada sería casi nula. En este ultimo caso los robot quedarían prácticamente quietos.

Las fórmulas adaptadas de las velocidades de cada rueda son las que se muestran en la ecuación 7.3.

$$\begin{array}{l} \upsilon_r = \frac{\upsilon_r}{\text{máx}(\upsilon_{lLV}, \upsilon_{rLV})} MaxVel \\ \upsilon_l = \frac{\upsilon_l}{\text{máx}(\upsilon_{lLV}, \upsilon_{rLV})} MaxVel \end{array} \tag{7.3}$$

Una observación importante es que los parámetros que indican la posición original del robot deben corregirse puesto que el método considera las coordenadas del centro del robot. En realidad lo que se pretende es que la parte delantera del robot llegue a un punto destino determinado. Esta corrección se basa en considerar el volumen del robot de modo que los parámetros utilizados en los cálculos del control LV son los que se muestran en la ecuación 7.4.

$$\begin{array}{ll} x_{LV} = x_r + \Delta x & , \Delta x = -1 \\ y_{LV} = y_r + \Delta y & , \Delta y = \left\{ \begin{array}{ll} 1,85 & si\, y_r \leq 0 \\ -1,85 & si\, y_r > 0 \end{array} \right. \end{array} \tag{7.4}$$

Siendo $(x_r,\, y_r)$ coordenadas reales.

- *Control Direct*

  Este control es una propuesta original del equipo FRUTo y surgió a raíz de la necesidad de contar con un control de movimiento simple pero de alta eficiencia.

  A diferencia del anterior, no toma en cuenta el ángulo de llegada y no asume ninguna hipótesis respecto a la posición relativa de los robots dentro del campo de juego.

  Las fórmulas de las velocidades de las ruedas son las que se muestran en la ecuación 7.5.

$$(\upsilon_r,\, \upsilon_l) = \left\{ \begin{array}{ll} (MaxVel + f(\alpha),\, MaxVel) & si\, \alpha < 0 \\ (MaxVel,\, MaxVel - f(\alpha)) & si\, \alpha \geq 0 \end{array} \right. \tag{7.5}$$

  Siendo f una función que convierte ángulos expresados en radianes a grados.

- *Control Direct LV*

  Es una combinación de los dos controles presentados anteriormente que resuelve el problema de ir desde un punto a otro llegando con cierto ángulo, superando así la limitante que presentaba el control LV (asume que la posición original del robot pertenece al cuadrante II o III de la cancha).

Se distinguen dos casos que se describen a continuación.

Caso 1: el robot se encuentra en el cuadrante II o III de la cancha. En este caso se aplica el control *LV* con las adaptaciones vistas. Ver figura 7.4.

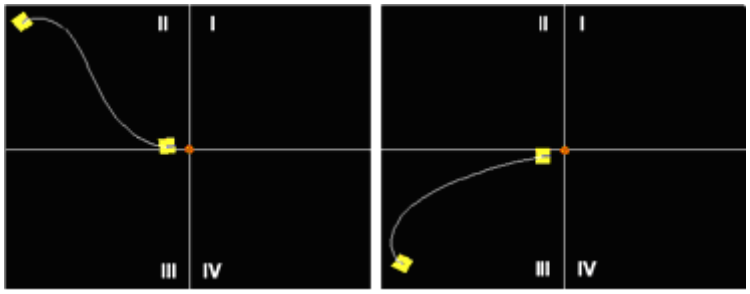


Figura 7.4: Control Direct LV - Caso 1.

Caso 2: el robot se encuentra en el cuadrante I o IV de la cancha. En este caso se aplica primero el control *Direct* para posicionar al robot en un punto del cuadrante II o III y a partir de esa posición se empieza a aplicar el control *LV* para llegar al destino con la orientación requerida. Ver figura 7.5.

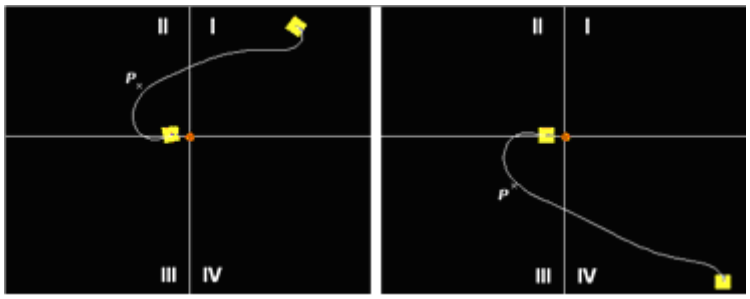


Figura 7.5: Control Direct LV - Caso 2.

- *Control Direct Ang*

  Básicamente, es igual al control *Direct* pero hace posible que los robots se desplacen sin tener en cuenta el sentido del cuerpo (mirando hacia adelante o hacia atrás) aprovechando la forma simétrica de los mismos.

  Este control se utiliza cuando el ángulo entre la orientación del robot y la orientación con la que se quiere llegar al destino es mayor que 90 grados, ya que evita tener que rotar el robot inicialmente antes de comenzar la trayectoria o al final antes de arribar al punto deseado.

  El cálculo de las velocidades se realiza como lo muestra la ecuación 7.6.

$$(v_r, v_l) = \begin{cases} (-MaxVel + f(\alpha), -MaxVel) & si\,\alpha < 0 \\ (-MaxVel, -MaxVel + f(\alpha)) & si\,\alpha \geq 0 \end{cases} \tag{7.6}$$

El beneficio aportado por este control en la rapidez para alcanzar el punto destino se muestra en la figura 7.6 .

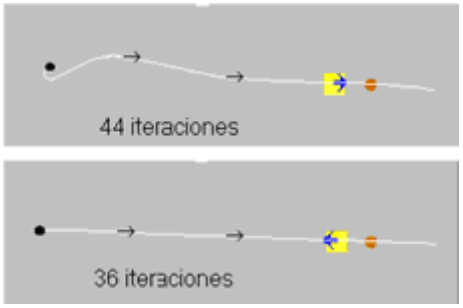


Figura 7.6: Las siguientes figuras muestran un robot partiendo de una posición con ángulo 180 grados y destino la pelota. Arriba, el control Direct primero rota el robot. Abajo, el control DirectAng llega marcha atrás.

- *Control Direct Ang2*

  Análogo al *Direct Ang* pero recibe un parámetro más que permite controlar la curvatura de la trayectoria.

- *Control CS (Splines Cúbicos)*

  A diferencia de los controles descritos anteriormente, este control permite contemplar el tiempo y velocidad de llegada al destino. Con este control es posible pasar del estado inicial $(x_\mathrm{i},\ y_\mathrm{i},\ r_\mathrm{i},\ t_\mathrm{i},\ |v_\mathrm{i}|)$ al estado final $(x_f,\ y_f,\ r_f,\ t_f,\ |v_f|)$.

  Para la aplicación de este control se consideran dos situaciones:

  - La situación estática, que maneja la hipótesis de que el punto destino no varía y por lo tanto se calcula la trayectoria una sola vez.
  - La situación dinámica, donde esa hipótesis no se cumple y por lo tanto la trayectoria debe recalcularse en cada iteración.

  En la práctica, la implementación de este control es parcial ya que, en algunas oportunidades, las aceleraciones lineales y/o angulares necesarias para seguir la trayectoria planificada exceden las capacidades de los robots. Por lo tanto la implementación no asegura el tiempo ni la velocidad llevando los robots desde estados iniciales de la forma $(x_i,\ y_i,\ r_i)$ hasta estados finales $(x_f,\ y_f,\ r_f)$.

  Además, al no tomar en cuenta la velocidad, se desarrolló un control de velocidades que permitiera disminuir la velocidad frente a curvas que, en la práctica, resultan imposibles de seguir a altas velocidades.

  En la figura 7.7 se muestran los resultados de una trayectoria con y sin control de velocidades:

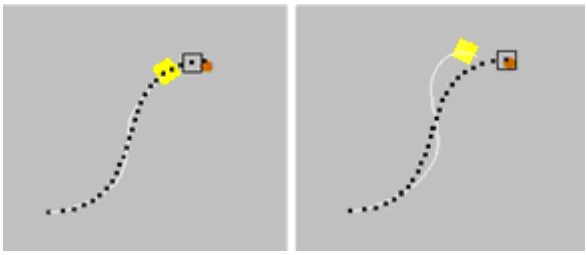


Figura 7.7: Trayectorias hechas por el control CS estático. Los cuadrados pequeños muestran las trayectorias halladas mediante splines cúbicas, el cuadrado grande representa el próximo punto significativo en la curva y la línea blanca es la trayectoria hecha por el robot. (a) muestra la solución final con control de velocidades, (b) muestra la solución sin control de velocidades.

7.1.3.2. Zonas libres

Las acciones combinables se valen de una herramienta para intentar generar situaciones de juego que puedan ser aprovechadas por otras acciones: el cálculo de *zonas libres* de obstáculos. Las zonas libres son áreas del campo de juego, libres de obstáculos, encerradas por circunferencias. El cálculo determina un conjunto de circunferencias candidatas a ser consideradas zonas libres y luego, para aquellas que se intersecan, selecciona la de mayor radio y descarta las demás. Esta herramienta apunta a convertir las zonas libres de obstáculos en objetivos atractivos para enviar la pelota (p. ej.: en caso de despejes).

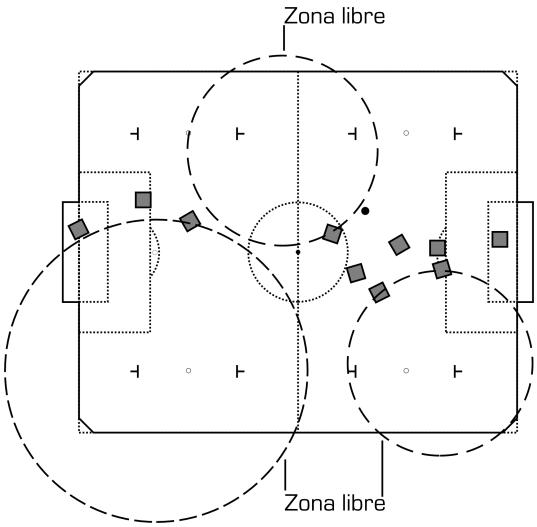


Figura 7.8: Cálculo de zonas libres..

7.1.3.3. Herramienta de cálculo de ángulos

Se utiliza para determinar la dirección de tiro al arco con mayor posibilidad de acierto o también para determinar a través de qué direcciones es más factible un pase hacia una zona libre de la cancha. Un ejemplo se puede apreciar en la figura 7.9.

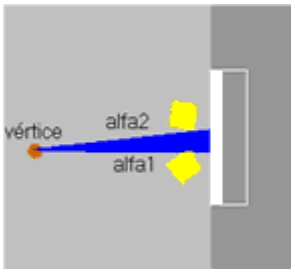


Figura 7.9: Herramienta auxiliar Ángulos

7.1.3.4. Predicción

Algunas acciones tienen como objetivo alcanzar la pelota para enviarla hacia un punto fijo, como el arco, o hacia un punto en movimiento, como el centro de una zona libre. Para que la acción sea eficaz, es necesario anticiparse de alguna forma al movimiento del objetivo. Para contemplar este problema, la solución propuesta realiza predicciones sobre la evolución de algunos aspectos del juego. La predicción es a corto plazo, ya que su fidelidad se deteriora al aumentar el tiempo considerado.

Para aquellas acciones relacionadas con alcanzar la pelota se realiza una proyección que permite aproximar la ubicación que tendrá el objetivo -la pelota- en las próximas iteraciones. En caso de que la acción pretenda enviar la pelota hacia una zona libre, se predice cómo evolucionarán la zonas libres en las próximas iteraciones. El control calcula las velocidades de las ruedas a partir de un objetivo calculado en función de la predicción de la posición de la pelota y, eventualmente, de la predicción de la posición del punto al cual se desea enviar la pelota.

Tanto la predicción de la evolución de las zonas libres como el cálculo del movimiento que permite esquivar obstáculos se basan en las predicciones de las trayectorias de todos los cuerpos móviles del juego: la pelota, los robots propios y los robots oponentes.

7.1.3.4.1. Predicción de la trayectoria de la pelota La predicción de la trayectoria de la pelota se calcula en función de su historia reciente, teniendo en cuenta su posición y velocidad lineal. El resultado está compuesto por la proyección de las posiciones que tendrá la pelota en las próximas iteraciones. Se tiene en cuenta la colisión con el perímetro de la cancha pero no con los robots. Para el modelo se considera que la pelota tiene aceleración constante en contacto con el terreno.

7.1.3.4.2. Predicción de la trayectoria de los robots La predicción de la trayectoria de los robots se calcula tanto para robots propios como para oponentes. Ambos cálculos se realizan para cada robot, en función de su historia reciente, teniendo en cuenta su posición, rotación y velocidades de las ruedas. El resultado está compuesto por la proyección de las posiciones que tendrá el robot en las próximas iteraciones. Como en la predicción de la trayectoria de la pelota, se tiene en cuenta la colisión con el perímetro de la cancha pero no con otros robots ni con la pelota. También se considera que los robots tienen aceleración constante sobre el terreno.

El modelo no pretende -por supuesto- predecir los cambios en las velocidades de los robots producto del comportamiento propio o del oponente. A efectos ilustrativos y a muy *grosso modo*, podría verse la predicción de la trayectoria de los robots como las posiciones que tendrían si, de pronto (en el momento en que se realiza la predicción), las estrategias se abstuvieran de enviar órdenes, quedando éstos influidos por la acción de una aceleración constante.

7.1.3.4.3. Predicción de la evolución de las zonas libres A partir de la predicción de la configuración futura de la pelota y de los robots propios y oponentes, se predice cuáles serán las zonas libres de obstáculos en las próximas iteraciones. El cálculo es idéntico al que se realiza para la configuración actual del juego, con la diferencia de que se utiliza la predicción de la configuración futura. La fidelidad del resultado depende, por lo tanto, de la fidelidad de las predicciones de las trayectorias de los cuerpos.

7.1.3.5. Evasión de obstáculos

La funcionalidad que otorga la habilidad de esquivar obstáculos puede activarse o desactivarse para una acción determinada, vía un mecanismo que permite configurar el sistema de manera *offline*. En caso de estar activada, una vez detectada una futura colisión con el robot al cual se le ha asignado la acción, se desvía la trayectoria para evitar el obstáculo. Esto se logra expropiando el control de la acción para redirigir el robot hacia una ruta alternativa que evite la colisión pero que mantenga el objetivo original (ver figura 7.2).

La detección de una futura colisión se fundamenta en la predicción de la trayectoria de la pelota y de los robots para las próximas iteraciones. Una vez detectada una posible colisión, se calcula un punto y una rotación intermedia para intentar desviar la trayectoria del robot de forma tal de evitar el obstáculo y llegar al destino original. La detección no contempla la posibilidad de que la nueva ruta trazada colisione con un nuevo obstáculo antes desechado. En caso de que esto ocurriera, se traslada la resolución a la próxima iteración.

El perímetro de la cancha no es considerado en la detección de obstáculos: si la trayectoria atravieza el perímetro, el robot terminará colisionando contra él. Para reducir el impacto negativo que esto pudiera tener sobre el desempeño del equipo, se intentan detectar y resolver posibles atascamientos de los robots.

Un ejemplo del funcionamiento de la evasión de obstáculos se muestra en la figura 7.10.

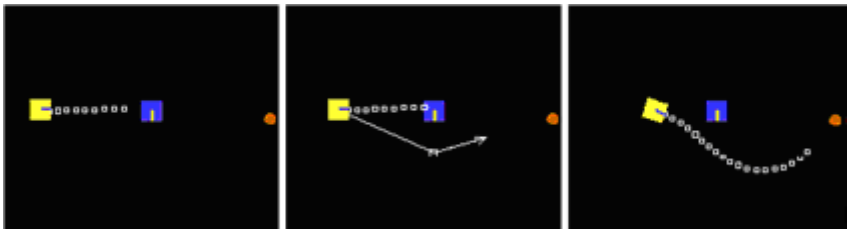


Figura 7.10: Evasión de obstáculos.

7.1.3.6. Manejo de atascamientos

La detección y resolución de atascamientos se realiza a nivel del planificador de movimientos. Se considera que un robot se encuentra atascado cuando se mantiene durante un intervalo de tiempo en una misma configuración por razones ajenas a él. La detección de un atascamiento depende de las velocidades asignadas al robot recientemente, así como de su configuración reciente: de manera abstracta, si durante un intervalo de tiempo arbitrario la configuración del robot no condice con la configuración que debería tener a partir de la asignación de velocidades, entonces se considera que el robot está atascado.

La resolución del atascamiento consiste en revertir el orden en que fueron asignadas las velocidades recientes, intercambiar los valores para la rueda izquierda y derecha y cambiarles su

signo (ver figura 7.11). Este manejo se basa en el supuesto de que fue el control quien provocó el atascamiento y que, por lo tanto, basta con invertir la trayectoria para salir de esa situación.

Figura 7.11: Control de Atascamiento. (a) el robot choca con la pared y queda atascado. (b) el control detecta el atascamiento e invierte los comandos enviados. (c) se reanuda el modo de funcionamiento normal.

7.2. PATITO Fútbol Club

7.2.1. Introducción

Esta sección se basa en la documentación presentada por este equipo en el congreso internacional FIRA Robot World Congress [Bis04b] y en el CAFR2004 [Bis04a]. No se ha encontrado más información que ésta, por lo que no se cuenta con detalles específicos de diseño e implementación. Se presenta brevemente las características principales del equipo desarrollado.

El desarrollo comenzó en febrero 2004, fue iterativo e incremental y las primeras tareas fueron analizar las reglas de la categoría SimuroSot así como las herramientas de desarrollo disponibles.

El principal objetivo que se planteó el grupo en esta primer etapa fue el de aprender el lenguaje Lingo y el uso del simulador SimuroSot. Para medir la competitividad del primer equipo desarrollado, se realizaron partidos de prueba con tres buenos equipos del CAFR 2003. Los resultados no fueron buenos en cuanto a los tanteadores finales de los partidos, pero la experiencia ganada a partir del estudio de las estrategias y documentos de esos equipos fue muy positiva.

El nuevo objetivo de grupo consistía en lograr un equipo de fútbol de robots competitivo, compuesto por cinco buenos jugadores, que heredaran de los equipos analizados las mejores características. El desarrollo del equipo de esta segunda fase comenzó por construir funciones básicas y herramientas necesarias para los jugadores. A partir de esas funciones básicas se construyeron luego las habilidades, que a su vez, se utilizaron para desarrollar los jugadores. Por último se desarrolló el equipo utilizando los jugadores construidos.

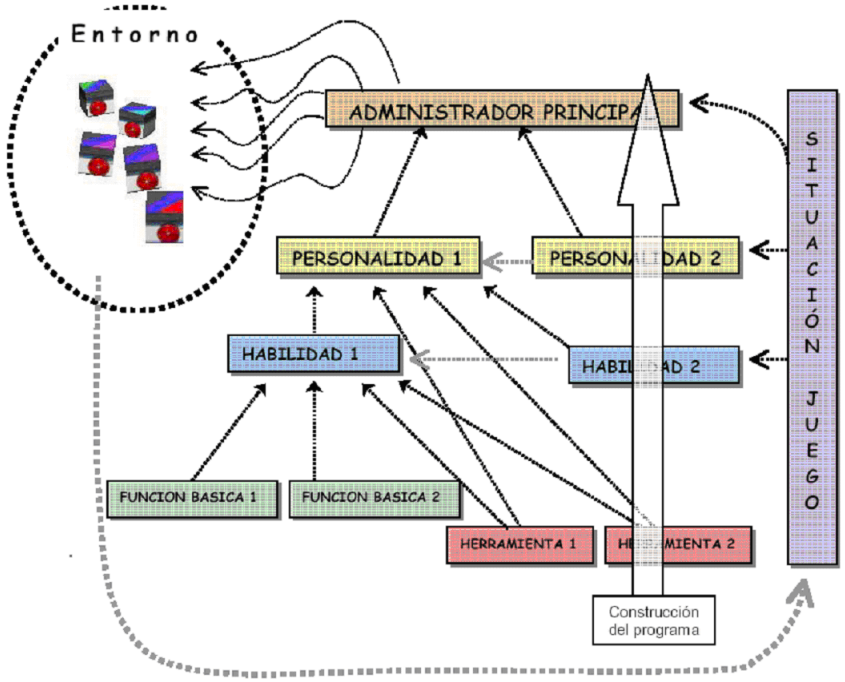


Figura 7.12: Funcionamiento y desarrollo del equipo Patito Fútbol Club. (Imagen extraída de [Bis04a])

El equipo Patito Fútbol Club consiste en cinco jugadores que son capaces de desempeñar todos los roles. Y según sus desarrolladores, la clave para ganar es: tener un equipo de cinco buenos jugadores individuales.

7.2.2. Ideas importantes

Cinco ideas importantes en las que se basa Patito Fútbol Club:

1. **Herramientas de cálculo**: Implementan las funciones que pueden predecir la posición futura de un objeto (robot o pelota) o la velocidad actual de estos objetos.

2. **Funciones primitivas**: Acciones básicas que cada robot es capaz de llevar a cabo. Esto involucra ir hacia un punto a una determinada velocidad, girar un ángulo determinado, rastrear la pelota, etc. Una característica destacada de estas herramientas es que permite a los robots moverse hacia adelante o hacia atrás indistintamente, permitiendo realizar movimientos en menos tiempo.

3. **Técnicas o Habilidades**: Uso de las herramientas de cálculo y las funciones primitivas para lograr acciones reales y más complejas como: patear, esperar, interceptar pelota, anticipar oponente, recuperar pelota, ir a un punto exacto.

4. **Personalidad de los Jugadores**: Utilizar todas las técnicas mencionadas anteriormente para que el robot sea un "jugador". Básicamente la táctica es despejar cuando se trata de

defensa, controlar más la pelota cuando se trata de juego en medio campo y atacar y ser más cuidadoso cuando se trata del ataque. El cometido general es mandar siempre la pelota hacia el campo contrario y preferentemente hacia el arco adversario. En total se desarrollaron diez personalidades, una completa [2]y otras más específicas para situaciones determinadas. Las personalidades se describen brevemente en la subsección 7.2.3.

5. **El Equipo**: Manejar la asignación de personalidades a los robots. Se cuenta con tres administradores que se encargan de la comunicación, coordinación y asignación de las personalidades a cada jugador según la situación de juego.

El equipo Patito Fútbol Club se basa en personalidades en lugar de basarse en roles de juego. Todos los jugadores son capaces de jugar en cualquier parte del campo de juego y adoptar la personalidad más conveniente según la situación.

7.2.3. Personalidades

Las personalidades posibles son diez y, excepto *FreePlayer,* casi todas son usadas para modelar los roles.

Todo el campo

- **FreePlayer:** Esta es la personalidad más completa. Es un jugador capaz de hacer cualquier tarea correctamente y jugar en todo el campo de juego. En todo momento habrá siempre un *FreePlayer*.

Ataque y medio campo

- **Soporte1:** Se posiciona siempre cerca del jugador que tiene la pelota (*FreePlayer*). Su tarea consiste en estar bien posicionado cuando sea necesario y cooperar con dicho jugador. Se establece una comunicación explícita entre ambos.
- **Soporte2:** A veces es necesario tener un segundo soporte que apoye al *Soporte1*.
- **Anotador:** Es un jugador "pescador" buscando siempre la oportunidad de convertir en el arco contrario. Esta posicionado esperando el pase de gol y nunca falla un gol fácil. Generalmente, esta personalidad anota la mitad de los goles del cuadro.
- **Medio-campista:** Da soporte a todos los ataques del cuadro. Es el primer defensa en los contra-ataques, recibe comunicación de todos los atacantes (por ejemplo: "perdimos la pelota") y actúa en consecuencia.

Defensa

- **ParedComun:** Es una tradicional defensa que se mueve por el eje Y, pero además despeja cuando considera oportuno.
- **ParedY:** Es un jugador que se mueve por el eje X , útil para defender los costados. Su característica es que despeja cuando considera oportuno y también sale jugando (cuando considera oportuno).
- **Defensor circular:** Circunda un lugar en su zona de gol. Generalmente, intercepta centros contrarios para salir jugando.

[2]Según la documentación del grupo la completitud de una personalidad aplica a la capacidad de desempeñarse en cualquier situación dentro del campo de juego.

- **Defensor libre:** Se mueve en un área libre de su zona de gol. Es un buen pateador de corners.
- **Golero:** Se comporta como golero y además interpreta el ambiente y actúa como sea necesario.
- **Golero Circular:** Comportamiento circular. Es usado la mayoría del tiempo menos para interceptar.
- **Ejecutor:** ejecuta penales y jugadas especiales. Cuenta con cuatro formas diferentes de ejecutar un penal, de modo que se pueda vulnerar a los diferentes estilos de arqueros.

El desarrollo del Jugador-Completo (*FreePlayer*) se llevó a cabo colocando a un aprendiz en el campo de juego y dejándolo jugar solo con la pelota. A medida que el jugador fue tomando mejor forma, se lo hizo jugar primero contra uno y luego contra dos oponentes menos inteligentes.

7.2.4. Coordinación

La coordinación del equipo se lleva a cabo a través de dos administradores que asignan las personalidades y otro que interpreta el ambiente.

El campo se divide en siete sectores y todo el entorno se interpreta en función de la pelota: posición, trayectoria y velocidad. No se fija en oponentes. Por otra parte, cada robot realiza un análisis del entorno según su personalidad y teniendo en cuenta sus compañeros y oponentes.

MainAdmin: controla a los cinco jugadores. En ataque, designará un par de atacantes, un *buscagol* [3] y un mediocampista. En defensa, designará un par de defensores, un jugador libre y un *buscagol*. Cuando suceden jugadas especiales asigna un ejecutor.

ComoAtacar: es el modulo que controla, asignando personalidades, al par de atacantes y su coordinación.

ComoDefender: es el modulo que controla, asignando personalidades, al par de defensores y su coordinación.

Situación Juego: divide al campo de juego en siete sectores, brinda información del juego en cuatro variables globales.

7.2.5. Comunicación

La comunicación entre los jugadores es explícita y de dos tipos: órdenes y anuncios. Las órdenes son dirigidas a robots específicos para lograr cooperación. Los anuncios generalmente son a todos los robots y comunican diferentes situaciones de juego.

Se definen jugadas predefinidas que generalmente se repiten mucho a lo largo de un juego y cada personalidad tiene un comportamiento por defecto para dichas jugadas.

7.2.6. Estructura

El comportamiento de las personalidades es elegido de un gran árbol de situaciones generales y un menú de situaciones particulares en el cual se consideran su propia interpretación y los anuncios de otros jugadores.

Cada personalidad puede adoptar otra personalidad cuando cree conveniente. El administrador asigna las personalidades, pero en algunos casos, esta personalidad "adopta" otra. Esto es una expresión de libertad. Puede que el jugador crea conveniente adoptar otra personalidad en base a su conocimiento del entorno (compañeros u oponentes).

[3] El término “buscagol”, más cocido como “pescador”, es utilizado por el equipo para denominar al jugador que permanece cerca del área oponente a la espera de una oportunidad de gol.

7.3. UBASot

7.3.1. Introducción

La propuesta del trabajo de tesis realizado en la UBA en el año 2002 por Castelo, Fassi y Scarpettini [CFS02] originalmente pretendía ser un relevamiento del estado del arte del fútbol de robots y de las diferentes técnicas utilizadas en los *MAS*. Luego de dicho relevamiento, se pretendía seleccionar e implementar algunas técnicas aplicables a robots móviles en una plataforma de simulación como paso inicial hacia la formación futura de un equipo de fútbol de robots que pudiera participar en los eventos internacionales como los organizados por la FIRA y RoboCup.

Una vez realizada dicha investigación se comenzó la implementación de comportamientos elementales de los robots, sobre la plataforma de simulación de RoboCup. Ya avanzada esta etapa se le presenta al grupo la posibilidad de que la Universidad de Buenos Aires participara en el Campeonato Mundial de Fútbol de Robots 2002 organizado por la FIRA, en Corea del Sur, a través del Grupo de Inteligencia Computacional Aplicada a Robótica Cooperativa en Sistemas Multirobots del Departamento de Computación de la Facultad de Ciencias Exactas y Naturales. El grupo no quiso dejar pasar la excelente oportunidad que se le presentó así que se avocaron al desarrollo de un equipo completo y competitivo en el marco de la categoría de simulación 5 vs 5 *Middle League SimuroSot* de la FIRA. El equipo desarrollado obtuvo el tercer puesto dentro de su categoría entre un total de cincuenta y nueve equipos participantes de doce países.

En esta sección se presentarán las consideraciones hechas por los integrantes del grupo y luego el diseño específico del equipo que implementaron para competir en la *Middle League SimuroSot*.

7.3.2. Características Generales

Se tuvieron en cuenta tres grandes puntos en el diseño del sistema o equipo de fútbol de robots:

- **Comprensión del entorno:** Se le otorgan ciertos sentidos a cada robot que le permiten comprender el entorno en el cual se mueve.
- **Decisiones tácticas y estratégicas:** En base a la información percibida del entorno se deben tomar decisiones y llevar a cabo acciones que vayan de acuerdo con la táctica del equipo. Se destacan tres puntos importantes: Aprendizaje, Estrategia y Modelado del oponente.
- **Ejecución de acciones:** Involucra las capacidades físicas del robot y la coordinación entre los agentes para poder llevar a cabo las acciones requeridas.

7.3.3. Aspectos de Diseño

El sistema se dividió en siete niveles que se corresponden con módulos o problemas a resolver referidos al diseño del sistema. Estos niveles son:

- Modelo del entorno
- Acciones
- Comportamientos elementales
- Comportamientos complejos
- Jugadas
- Tácticas
- Estrategias

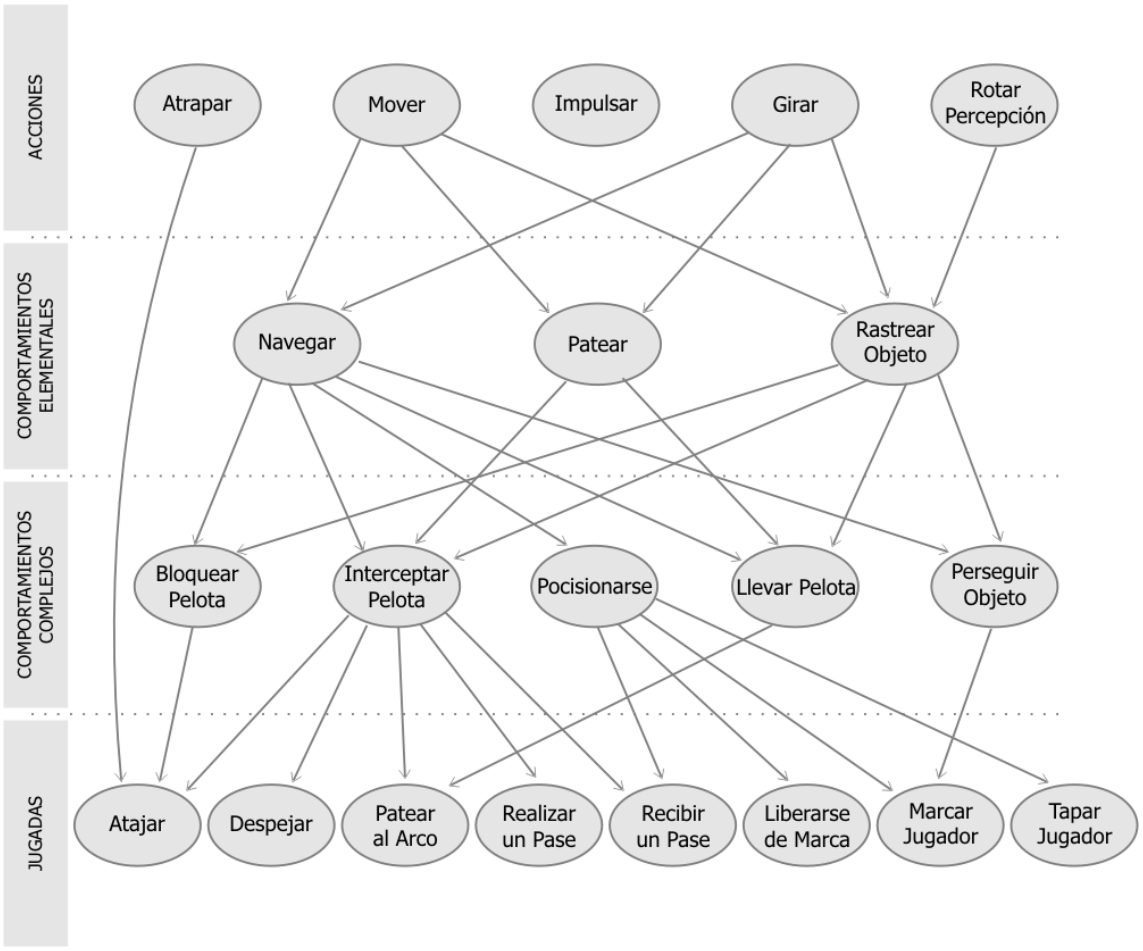


Figura 7.13: Niveles definidos en el análisis del equipo UBASot.

La figura 7.13 muestra la relación entre alguno de los diferentes niveles y sus funcionalidades.

A continuación se presentan los puntos más relevantes de cada uno de los niveles.

Modelo del Entorno

El primer elemento a identificar en el entorno es el robot mismo (autolocalización).

Conociendo el modelo físico actual de los objetos, se pueden predecir estados futuros o completar la información presente en base a estados pasados.

Consideraciones sobre el modelo del entorno:

- **Sistemas de percepción:** Obtener información del ambiente a partir de la percepción.
- **Autolocalización:** Ubicarse geográficamente dentro del campo de juego.
- **Reconstrucción del estado global:** Obtener un estado completo del entorno (más complejo en sistemas con visión local como en RoboCup, que en sistemas con visión global de la FIRA).

- **Modelo de objetos:** Definir un modelo físico de los objetos dentro del ambiente en el que se desarrolla el juego.
- **Predicción:** Obtener un modelo del entorno en un tiempo futuro, o actual en base al pasado.

Acciones

En el caso de robots reales, los actuadores del robot determinarán la forma de implementación de las acciones en gran medida. No será lo mismo moverse con ruedas que con dos o cuatro patas o apéndices.

En el caso de un entorno simulado sucede algo similar. Algunos simuladores ofrecen un conjunto de acciones nativas que el jugador posee mientras que en otros cada una de estas acciones requieren un desarrollo específico.

Acciones comunes en fútbol de robots identificadas por UBASot:

- **Mover:** Desplazar al agente sobre la superficie del campo de juego.
- **Girar:** Cambiar de orientación sin desplazarse de su posición actual.
- **Impulsar:** Separar un objeto del robot mediante un impulso.
- **Atrapar:** Retener un objeto cerca del robot.
- **Rotar percepción:** Cambiar el ángulo de la percepción sin cambiar la orientación del robot.

Comportamientos Elementales

Se definen comportamientos elementales en fútbol de robots a aquellos comportamientos que representan el primer contacto con el entorno y que sirven de base para el desarrollo de comportamientos más complejos.

Los comportamientos elementales en fútbol de robots, tomados en cuenta por UBASot, son:

- **Navegar:** Moverse de un lugar a otro teniendo en cuenta el entorno.
- **Patear:** Impulsar la pelota hacia un determinado lugar, generalmente el arco contrario o hacia un compañero.
- **Rastrear un objeto:** Seguir el movimiento de un objeto sin moverse del lugar ,de manera de no perderlo de vista.

Para implementar cada comportamiento elemental pueden utilizarse distintas técnicas.

En algunos casos se utiliza una estrategia estática y determinista, mientras que para otros puede utilizarse algún algoritmo de optimización o incluso utilizar técnicas de aprendizaje.

Una forma de utilizar aprendizaje por refuerzo podría ser intentar predecir el estado del juego t instantes de tiempo posteriores al actual. Se observan las distintas posibilidades futuras y se clasifican como positivas o negativas y se suman al conocimiento del robot.

Se toma la mejor decisión en el momento y luego se incorpora el resultado del movimiento (positivo o negativo) al conocimiento del robot, para que en movimientos posteriores puedan ser tenidos en cuenta a la hora de decidir hacia donde moverse.

El aprendizaje significa mejorar incrementalmente la política de decisión de manera tal que el objetivo se cumpla de una forma cada vez más eficiente.

Comportamientos Complejos

En base a los comportamientos elementales vistos anteriormente, podemos obtener comportamientos más complejos que darán lugar luego a las jugadas dentro del campo de juego. Los comportamientos complejos se obtienen a partir de la composición de los comportamientos elementales.

A continuación se enumeran los comportamientos complejos detectados:

- **Interceptar la pelota:** Desplazarse de forma de colisionar la pelota en movimiento.
- **Bloquear la pelota:** Ubicarse en algún punto futuro de la trayectoria de la pelota.
- **Posicionarse:** Elegir e ir hacia un determinado punto de la cancha.
- **Llevar la pelota:** Desplazarse de un lugar a otro sin perder contacto con la pelota.
- **Perseguir un objeto:** Mantenerse cerca de un objeto móvil conservando la distancia.

Jugadas

Las jugadas son comportamientos de alto nivel que combinan los comportamientos complejos para poder realizarse.

Algunas de las jugadas más usadas en fútbol de robots y relevadas por UBASot son:

- **Atajar:** Acción del arquero tendiente a evitar que la pelota ingrese al arco.
- **Despejar la pelota:** Impulsar la pelota fuera del área defensiva reduciendo el peligro para el arco propio.
- **Patear al arco:** Intentar convertir un gol.
- **Realizar un pase:** Enviar la pelota a una posición de forma tal que quede bajo el control de un robot compañero.
- **Recibir un pase:** Posicionarse para recibir el pase de un compañero.
- **Marcar a un jugador:** Ubicarse cerca de otro jugador de forma de anticiparse a sus movimientos y evitar que tome control de la pelota o que la patee al arco propio.
- **Liberarse de una marca:** Ubicarse de forma que los defensores del equipo contrario no interfieran en posibles recepciones de la pelota o de tiros al arco.
- **Tapar un jugador:** Molestar a un robot para que no pueda tomar control de la pelota.

Tácticas

La problemática de este nivel se relaciona básicamente con el comportamiento de cada robot según el rol o puesto que le toca dentro del equipo y, en base a esto, las decisiones que debe tomar en cada momento respecto a su comportamiento. Por ejemplo, determinar con qué compañero cooperar, a qué rival marcar, dónde ubicarse, etc.

Estrategias

La estrategia no es más que el plan que lleva a cabo el equipo durante el partido: la distribución de sus jugadores en el campo, el rol que desempeña cada uno, si éste lo mantiene durante todo el partido o cambia de acuerdo a las circunstancias, los códigos para la comunicación, las jugadas pre-armadas, los comportamientos individuales, etc.

7.3.4. Política de juego

A continuación se describen los elementos que se consideraron determinantes para definir variaciones en la política de juego del equipo UBASot:

- **Resultado parcial del partido:** Una diferencia importante, tanto positiva como negativa, puede inducir a cambiar el tipo de estrategia del equipo.
- **Tiempo de juego restante:** En muchas ocasiones este dato se complementa con el anterior. Por ejemplo, si el resultado es favorable y resta poco tiempo de juego es posible que se adopte un estilo de juego defensivo y lento. En cambio, si resta poco tiempo y el equipo está perdiendo el partido, es muy probable que el equipo se adelante para conseguir goles.
- **Estrategia del equipo oponente:** El equipo puede contar con información almacenada sobre el estilo de juego de su oponente y variar su estrategia consultando estos datos. También puede contar con mecanismos para realizar un estudio *online* de su rival y actuar en consecuencia.
- **Información estadística:** También se puede optar por la alternativa de recopilar información propia y de equipos rivales y generar una base de información estadística a ser considerada durante la toma de decisiones. Esta base puede ser creada previamente y enriquecida durante el transcurso de cada partido (aprendizaje *online*).

7.3.5. Modelado del Oponente

El modelado del equipo oponente se abordó mediante tres enfoques. Los mismos pueden ser combinados o aplicados individualmente.

- **Análisis de partidos jugados:** Se trata de recopilar partidos jugados por el equipo adversario y, mediante la observación de un experto, determinar las fortalezas y debilidades del equipo evaluado.
- **Reconocimiento de estrategias:** Esta alternativa consiste en observar el juego desde afuera y descubrir las estrategias de alto nivel empleadas por el adversario. Esta función la cumple un *coach* y se realiza en tiempo real, a diferencia del análisis planteado en el enfoque anterior.
- **Rastreo:** Se trata de un seguimiento dinámico del oponente, sus intenciones y objetivos, para predecir su juego y actuar en consecuencia. El desafío consiste en rastrear al adversario en tiempo real a pesar de las ambigüedades, y rastrear al equipo más que a cada jugador en forma individual.

7.3.6. Ejemplos de Estrategias Aplicadas

Se consideraron tres esquemas en cuanto a la coordinación del equipo:

- **Coordinación de agente externo:** Aparece la figura del *coach*. Se cuenta con un módulo auxiliar que se encarga de realizar las observaciones del partido, procesar dicha información conjuntamente con información estadística y del entorno, y obtener un resultado que es comunicado a cada jugador. Es necesario contar con mecanismos de comunicación entre dicho módulo *coach* y el resto de los jugadores.
- **Coordinación de sub-equipos:** El equipo se subdivide en grupos, donde cada grupo es un sub-equipo con un líder asignado. Es el líder de cada sub-equipo quien conoce el entorno, analiza la información y comunica luego a los jugadores de su subgrupo la estrategia a aplicar. Al igual que en el caso anterior es necesario contar con algún sistema de comunicación entre los jugadores del cada subgrupo.

- **Coordinación implícita:** En sistemas donde no se cuenta con comunicación entre los jugadores no es posible utilizar la figura de *coach* o líderes de subgrupos. En estos casos se utiliza la comunicación implícita, donde cada jugador tiene conocimiento de la estrategia de los demás jugadores porque conoce implícitamente a los mismos (a los de su mismo rol), y de esta forma puede predecir su comportamiento frente a determinadas circunstancias.

7.3.7. Implementación del equipo UBASot 2.10

El equipo se desarrolló sobre la plataforma del simulador *Middle League SimuroSot* de la FIRA.

Como lineamientos generales, el equipo UBASot optó por trabajar con roles, formación fija, modelado sencillo del movimiento del oponente y sin comunicación explícita entre los jugadores.

Con respecto al estilo de juego, las premisas son:

- Defensa sólida del arco propio mediante un arquero y defensas eficaces.
- Atacar el arco rival siempre que sea posible, priorizando velocidad sobre precisión.

Por otro lado se intenta mantener la pelota en movimiento la mayor parte del tiempo, aunque esto pueda comprometer la precisión, además de hacer lo posible por llevar la pelota al campo rival. La implementación de los movimientos busca la mayor velocidad posible en los robots, asumiendo el costo de ocasionar colisiones o mayor imprecisión. Los robots son ubicados en posiciones estratégicas, evitando que interfirieran en jugadas donde no participan.

Para obtener una respuesta rápida se simplificaron al máximo los cálculos relacionados con el modelado para asegurar respuestas en tiempo.

7.3.8. Arquitectura

De la arquitectura del sistema se destacan tres componentes:

- **Ambiente:** Es el encargado de registrar y suministrar información sobre el estado del juego. Este módulo almacena estados de juego anteriores para realizar predicciones y modelar el comportamiento del equipo rival.
- **Control:** Es el responsable de proveer las acciones de los jugadores, básicamente las relacionadas con el pateo, la navegación y movimientos de cada robot. También lleva registro de cuál es el módulo de Rol que controla cada robot en cada momento.
- **Roles:** Son responsables del perfil de juego de cada jugador. Se implementaron tres diferentes: Arquero, Defensor y Delantero. La asignación de roles es estática y no cambia a lo largo del juego. El equipo está conformado por un arquero, dos defensores y dos delanteros.

A su vez, estos tres componentes dependen de un módulo Principal. Dicho módulo es el encargado de la comunicación con el simulador y administrar a los demás módulos y el juego de equipo.

La figura 7.14 muestra la estructura general del sistema diseñado:

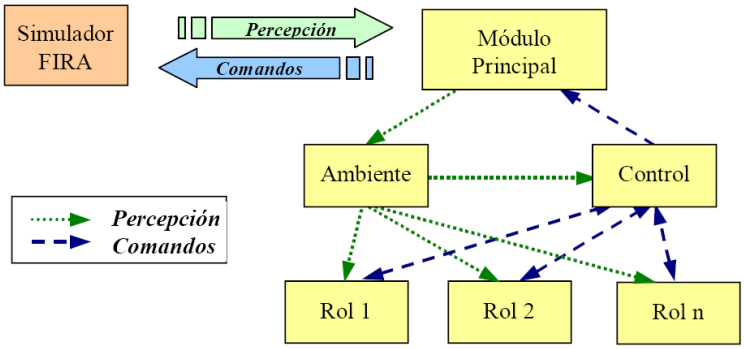


Figura 7.14: Arquitectura del equipo UBASot. (Imagen extraída de [CFS02])

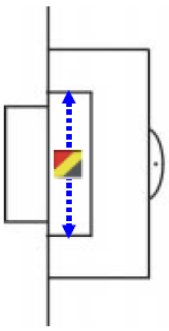


Figura 7.15: Movimientos del Arquero. (Imagen extraída de [CFS02])

7.3.9. Roles

El arquero siempre realiza movimientos paralelos al arco, variando solo su coordenada Y para desplazarse e interceptar la pelota evitando el gol contrario.

La figura 7.15 muestra el desplazamiento del arquero.

En el caso de los defensas, uno es el defensa principal y el otro el secundario, e intentan evitar que la pelota llegue al arco cuando se encuentra en posesión del rival.

Se realizan dos tipos de defensa:

- **Activa:** Ir al encuentro de la pelota ya sea para despejarla o para obstruir su trayectoria.
- **Pasiva:** Posicionarse en una zona estratégica para evitar el pasaje de los rivales.

Cuando el defensor principal se dirige hacia la pelota para interceptarla, el defensa secundario cubre la posición complementando así al primero.

Los dos robots que cumplen el rol de delanteros se ubican generalmente en campo rival y se encargan de convertir goles. También se define siempre un delantero principal y uno secundario. Cuando el delantero principal posee la pelota y ataca al arco, el delantero secundario se ubica de forma tal de quedar alerta ante posibles pases o rebotes. Por lo general se intenta que los delanteros no se acerquen demasiado al área propia para no estorbar en las actividades de defensa. Cuando

la pelota se encuentra en el área grande del rival, ambos delanteros son principales y se lanzan al ataque, para tener así una mayor ofensiva.

7.3.10. Comportamientos y Jugadas

El equipo UBASot implementó acciones especiales para jugadas de pelota quieta. Las acciones especiales son:

- Navegar
- Ir a una zona
- Parar
- Girar
- Patear recto
- Patear de costado
- Patear girando
- Patear al arco

Estos comportamientos se disparan según el rol que posea el robot y el estado de juego.

Para la predicción se utilizó una estructura auxiliar interna donde se almacena información sobre la posición de los objetos en la cancha (pelota y jugadores).

El movimiento de la pelota se modeló linealmente, sin tener en cuenta trayectorias curvas. Se tomó como despreciable el rozamiento que puede afectar a la pelota, y por tanto, no se tomó en cuenta dicho caso para calcular posiciones futuras de la misma. Además el tiempo de predicción a futuro es pequeño, como para que dicha fuerza realmente llegue a afectar la velocidad de la pelota.

La fórmula utilizada para la proyección futura de la pelota se muestra en la ecuación 7.7.

$$\begin{aligned} x_{t+i} &= x_t + \Delta x * i \, donde \, \Delta x = x_t - x_{t-1} \\ y_{t+i} &= y_t + \Delta y * i \, donde \, \Delta y = y_t - y_{t-1} \end{aligned} \tag{7.7}$$

Si la proyección termina fuera de la cancha, se retorna el borde de la misma como punto final de la trayectoria.

Para el caso de los robots, la mayoría de sus movimientos describen arcos, debido a la característica de poseer dos ruedas laterales independientes. En este caso se debe tener en cuenta la velocidad angular y el radio del círculo que describe para poder calcular la trayectoria futura.

La fórmula 7.8 describe dicha proyección:

$$\Delta x_i = \frac{\sqrt{\Delta x^2 + \Delta y^2}}{Va} * (\sin(\alpha_t - Va * \Delta t) - \sin(\alpha_t + i * Va * \Delta t))$$

$$\Delta y_i = \frac{\sqrt{\Delta x^2 + \Delta y^2}}{Va} * (-\cos(\alpha_t - Va * \Delta t) + \cos(\alpha_t + i * Va * \Delta t))$$

$$\Delta \alpha_i = Va * i$$

$$y_{t+i} = y_t \pm \Delta y_i$$
$$x_{t+i} = x_t \pm \Delta x_i$$

$$\alpha_{t+i} = \alpha_t \pm \Delta\alpha_i \tag{7.8}$$

Para la navegación se utilizó un algoritmo heurístico, que detecta en cada ciclo de simulación el mejor camino a tomar para llegar a la posición final proyectada. Este intenta buscar trayectorias casi lineales en cada momento evitando las colisiones. En todos los movimientos se tiene en cuenta la inercia de los objetos según la velocidad con la que se mueven. Dicha inercia puede tomar valores entre 0 y 4 según la velocidad que posea el objeto.

Al momento de patear siempre se busca un movimiento que permita dirigir la pelota hacia el arco rival, teniendo en cuenta el movimiento y la posición actual, la inercia y los obstáculos (robots o paredes de la cancha). No siempre se tiene en cuenta el punto medio del robot para determinar el movimiento y punto de impacto futuro del mismo con la pelota. Por ejemplo, para patear al arco podría tenerse mejor ángulo de pateo si se impacta la pelota sobre la esquina izquierda de la cara frontal del robot que con el punto medio. Eso se muestra en la figura 7.16.

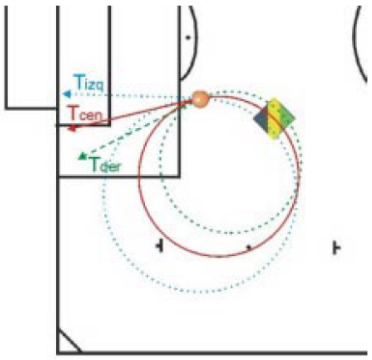


Figura 7.16: Tiro al arco. (Imagen extraída de [CFS02])

Una vez calculada la mejor trayectoria, se le asigna a las ruedas la potencia necesaria para que el robot describa el movimiento deseado.

Las fórmulas 7.9 y 7.10 muestran el cálculo realizado para determinar la potencia, donde EJE es la longitud del eje de ruedas del robot:

$$potenciaExterna = potenciaMaxima \tag{7.9}$$

$$potenciaInterna = potenciaExterna * \frac{Radio - \frac{EJE}{2}}{Radio + \frac{EJE}{2}} \tag{7.10}$$

El comportamiento patear de costado se utilizó básicamente en jugadas en las que la pelota toma contacto con uno de los laterales del robot. En este caso se realiza un giro en el lugar (máxima potencia opuesta en ambas ruedas) para que la pelota se despegue del robot. Se tomó como caso especial cuando la trayectoria resultante de la pelota puede finalizar en el arco propio.

El equipo UBASot implementó algunas acciones especiales a llevarse a cabo en jugadas de pelota quieta. Las jugadas especiales de pelota quieta son:

- **Saque del centro de la cancha:** El robot en campo contrario más cercano a la pelota, patea hacia el campo propio. Los demás robots se detienen hasta que la pelota se pone en movimiento.

- **Saque de arco:** El defensa más cercano a la pelota realiza el tiro. El arquero cubre siempre el arco.
- **Pique:** El robot con la coordenada Y más cercana a la pelota realiza el tiro. Si está en dirección a su arco desvía la pelota hacia los laterales, si está en dirección al arco contrario patea hacia el centro de la cancha.
- **Tiro libre:** Siempre se intenta patear al arco rival.
- **Penal:** Lo ejecuta el robot más cercano a la pelota. Cuando se debe atajar el arquero se ubica en la misma coordenada Y que la pelota.

Bibliografía

[ATB04] John Andreson, Brian Tunner, and Jacky Baltes, *Reinforcement learning from teammates of varying skill in robotic soccer*, FIRA Robot World Congress (2004), Autonomous Agents Laboratory, Departament of Computer Science, University of Manitoba, Canada. 29

[Bis04a] Daniel Bistman, *Patito fútbol club*, CAFR (2004). 9, 70, 71

[Bis04b] _____, *Patito fútbol club, how this program works?. how these robots think?*, FIRA Robot World Congress (2004). 70

[Bit05] Guilherme Bittencourt, *Introdução aos sistemas multiagentes - histórico e fundamentos*, Brazil Agents School, SBC, Sep 2005. 16

[BW03] David Ball and Gordon Wyeth, *Clssifying an opponent's behaviour in robot soccer*. 55

[CC04] Alvaro Castromán and Ernesto Copello, *Fútbol de robots uruguayo para torneos: el equipo*, Tech. report, Instituto de Computación, Facultad de Ingeniería, Universidad de la República, Uruguay, Mayo 2004. 57, 63

[CDB96] A. Collinot*, A. Drogoul*, and P. Benhamou**, *Agent oriented design of a soccer robot team*, ICMAS'96 Workshop on Animal Societies and DAI, Kyoto, Japan (1996), *LAFORIA, University of Paris, Francia. **National Institute for Aerospace Research and Studies, Châtillon, Francia. 26

[CFS02] Claudia C. Castelo, Héctor R. Fassi, and Flavio E. Scapettini, *Fútbol de robots: Revisión del estado del arte y desarrollo del equipo ubasot de simulación*, Tech. report, Departamento de Computación, Facultad de Ciencias Exactas y Naturales, Universidad de Buenos Aires, Buenos Aires, Argentina, 2002. 10, 74, 80, 82

[CFV04a] Camilo Cerchiari, Javier Frank, and Martín Varela, *Agentes inteligentes, estado del arte*, Tech. report, Instituto de Computación, Facultad de Ingeniería, Universidad de la República, Uruguay, 2004. 23, 29, 35, 36, 37, 39

[CFV04b] _____, *Agentes inteligentes, reporte final*, Tech. report, Instituto de Computación, Facultad de Ingeniería, Universidad de la República, Uruguay, 2004. 24

[DH73] Richard Duda and Peter Hart, *Pattern classification and scene analysis*, John Wiley & Sons Inc, Jun 1973, ISBN: 0471223611. 55

[DS04] Marco Dorigo and Thomas Stützle, *Ant colony optimization*, The MIT Press, Julio 2004, ISBN-10: 0-262-04219-3, ISBN-13: 978-0-262-04219-2. 49

[dTdI06] GTI Grupo de Tratamiento de Imágenes, *Sistemas de reconocimiento de patrones*, Instituto de Ingeniería Eléctrica, Facultad de Ingeniería, Universidad de la República, Uruguay, 2006. 53, 54, 55

[FG96] Stan Franklin and Art Graesser, *Is it an agent, or just a program? a taxonomy for autonomous agents*, Third International Workshop on Agent Theories, Architectures and Languages, Springer-Verlag (1996), Institute for Intelligent Systems, University of Menphis, USA. 15

[FS91] James A. Freeman and David M. Skapura, *Neural networks: algorithms, applications and programming techniques*, Addison-Wesley, 1991, ISBN: 0-201-51376-5. 20

[Hüb05] Jomi Fred Hübner, *Especificaçao organizacional de sistemas multiagentes e o modelo moise+*, Brazil Agents School, SBC, Sep 2005. 47

[HG02] William H. Hsu* and Steven M. Gustafson**, *Genetic programming and multi-agent layered learning by reinforcements*, Genetic and Evolutionary Computation Conference, New York, USA (2002), *Department of Computing and Information Sciences, Kansas State University, USA. **School of Computer Science and Information Technology, University of Nottingham, UK. 44

[KLM96] Leslie Pack Kaelbling*, Michael L. Littman*, and Andrew W. Moore**, *Reinforcement learning: A survey*, Journal of Artificial Intelligence Research (1996), *Computer Science Department, Brown University, Providence, RI, USA. **Carnegie Mellon University, Pittsburgh, PA, USA. 23, 27, 28

[KvdS96] Ben Krose and Patrick van der Smagt, *An introduction to neural networks*, 8th ed., The University of Amsterdam, Nov 1996. 20

[LF04] Tomás Laurenzo and Gabrielle Facciolo, *Una herramienta de análisis de estrategias de fútbol de robots middle league simurosot.* 9, 56

[Luk98] Sean Luke, *Genetic programming produced competitive soccer softbot teams for robocup97*, The Second International Workshop on RoboCup (1998), Department of Computer Science, University of Maryland, USA. 41, 42, 44

[PAC04] Vasco Pires, Miguel Arroz, and Luis Custódio, *Logic based hybrid decision system for a multi-robot team*, In Proceedings of the Eighth Conference on Intelligent Autonomous Systems, Mar 2004, Institute for Systems and Robotics, Lisboa, Portugal. 16, 17

[PP02] Sankar Pal and Amita Pal, *Pattern recognition, from classical to modern approaches*, World Scientific Publishing Company, Jan 2002, ISBN: 9810246846. 54

[PZ04] Chris A. C. Parker and Hong Zhang, *Biologically inspired decision making for collective robotic systems*, IEEE/RSJ International Conference on Intelligent Robots and Systems (2004), Dept. of Computing Science, University of Alberta, Alberta, Canada. 50, 52

[RN02] Stuart Russell and Peter Norvig, *Artificial intelligence: A modern approach*, segunda ed., Prentice Hall, Dec 2002, ISBN: 0137903952. 55

[SB04] Richard S. Sutton and Andrew G. Barto, *Reinforcement learning i: Introduction*, FIRA Robot World Congress (2004). 25, 26

[SK00] William D. Smart* and Leslie Pack Kaelbling**, *Practical reinforcement learning in continuous spaces*, FIRA Robot World Congress (2000), *Computer Science Departament, Brown University, USA. **Artificial Intelligence Laboratory, MIT, USA. 30, 31