UNIVERSIDAD NACIONAL DE COLOMBIA

Simulador de fútbol de Robots

Fabio Fabian Barbosa Calvo

Universidad Nacional de Colombia
Facultad de Ingeniería, Departamento de Ingeniería de Sistemas e Industrial
Bogotá, Colombia
2016

Simulador de fútbol de Robots

Fabio Fabian Barbosa Calvo

Trabajo Final presentado como requisito parcial para optar al título de:
Magister en Ingeniería de Sistemas y Computación

Director:
Ph.D., Master Jonatan Gómez Perdomo

Línea de Investigación:
Ecología, Sociedad y Cultura Artificial
Grupo de Investigación:
ALIFE

Universidad Nacional de Colombia
Departamento de Ing. de Sistemas e Industrial
Bogotá, Colombia
2016

Agradecimientos

Un agradecimiento muy especial merece la comprensión, paciencia y el ánimo recibidos de mi familia y amigos.

Agradezco al profesor Jonatan Gómez Perdomo, por su enorme paciencia y orientación en la misma.

Agradezco a mis compañeros y amigos Rodrigo Moreno, Arles Rodriguez, Camilo Cubides, Leonardo Quiñonez y todos los demás integrantes del grupo ALife, por su colaboración en la realización del presente trabajo.

Resumen

En este trabajo se propone y desarrolla un ambiente simulado (FuroSimSpark), ambiente que permite la coordinación y sincronía del comportamiento de un conjunto de robots físicos futbolistas antes, durante y después de la competencia. El fútbol de robots es el campo de la robótica colaborativa que permite la experimentación de diversos comportamientos en sistemas multi-agentes (Robots). Estos comportamientos se programan en dichos robots, para que actúen de forma independiente y grupal, aprendan del juego, solucionen problemas y creen estrategias que les permitan encontrar y llevar el balón a la meta rival para hacer goles y evitar que el equipo oponente cumpla este objetivo.

Palabras clave: Robot, Simulador, Multi-agentes

Contenido

Lista de figuras

Lista de tablas

Introducción

La robótica móvil es la parte de la robótica que se centra en el desarrollo de robots con capacidad de desplazarse en un ambiente [38]. El desarrollo de esta tecnología implica grandes retos en mecánica, electrónica e inteligencia artificial [38]. Una de las formas de promover estas áreas es mediante juegos como el fútbol de robots. El fútbol de robots es una aplicación de la robótica móvil, en la cual un equipo de robots debe ser capaz de explorar un ambiente determinado (campo de fútbol) para encontrar el balón, llevarlo a la meta rival, hacer goles y evitar que el equipo oponente cumpla dichas tareas [1]. Desde mediados de los años 90, se realizan diversas competencias de fútbol de robots, entre las que se encuentra RoboCup (Robot Soccer World Cup) [2] y FIRA (Federation of Robot-soccer Asociations) [3].

Para cumplir con la tarea, que tiene un equipo de fútbol de robots, se requiere el desarrollo de estrategias de coordinación, control y diseño, tanto de la parte física de los robots como de las estrategias de coordinación. Para el desarrollo de estas estrategias se usan a menudo robots simulados. El uso de simuladores como apoyo en la robótica no es algo nuevo [32]. Para agilizar el desarrollo del software se usan herramientas de simulación, ya sea de desarrollo propio o los que se encuentran disponibles públicamente. Muchos de estos sistemas han evolucionado para simular las características más importantes del mundo real [32].

El objetivo de este trabajo es, por un lado, proveer un marco teórico que sirva para futuros trabajos y desarrollos en el área y por el otro presentar el proceso de creación del simulador llamado FuroSimSpark, en el cual se puede simular el

fútbol de robot. FuroSimSpark proporciona un entorno multi-agente en el que se puede ver el desempeño y desarrollo estratégico de estos sistemas, habilidades tales como movilización hacia un objetivo determinado, habilidad para llevar la pelota a través de un recorrido, evitar obstáculos móviles, golpear la pelota al arco, ubicarse dentro del campo de juego, evaluar su posición con respecto a sus compañeros y oponentes, interceptar la pelota en movimiento y compartir una estrategia de equipo.

El simulador FuroSimSpark proporciona una imagen aproximada del ambiente y de los sistemas físicos en tiempo real, tales como las dinámicas de los cuerpos rígidos y la detección de colisiones con el uso del motor físico ODE (Open Dynamics Engine™) [17]. La dinámica de estos elementos les permite a los robots interactuar con el entorno y con los demás agentes.

En particular, en este trabajo se desarrolla un software que simula los elementos básicos del fútbol de robots (cancha, jugadores y balón), se programan las estrategias que siguen los jugadores en diferentes escenarios como son el comportamiento de sus oponentes, el comportamiento cooperativo grupal y la navegabilidad emulada que debe ser lo más cercano al entorno real. El simulador FuroSimSpark permite continuar con el diseño, desarrollo e implementación de un equipo de fútbol robótico, en el que la etapa de arquitectura y construcción de los sistemas multi-agente tiene un alto grado de maduración y que junto con la implementación del simulador desarrollado para este trabajo, afianza y genera nuevas fuentes de investigación.

Este trabajo presenta en su primer capítulo los antecedentes en las áreas de robótica, inteligencia artificial y simulación así como su aporte al desarrollo tecnológico. El segundo capítulo comprende la introducción al entorno de fútbol de robots, los problemas que este entorno presenta y las características de los sistemas de simulación. El Capítulo tres describe la implementación física de los robots realizada en el grupo de investigación Alife, y el funcionamiento del Simulador FuroSimSpark. Finalmente se presentan los resultados y conclusiones obtenidas a partir de los ejercicios realizados.

1. Antecedentes

Este capítulo tiene como uno de sus objetivos explicar las generalidades de la robótica, la robótica móvil y los simuladores. Además otro de los objetivos de este capítulo es mostrar cómo la evolución de estas áreas, ha permitido desarrollar grandes avances tecnológicos y científicos en la industria. Finalmente también describe la importancia de la inteligencia artificial como apoyo a la autonomía y coordinación de los robots.

1.1 Robótica

Según La Sociedad de Robótica y Automatización de la IEEE (Robotics and Automation Society, IEEE), la robótica es una ciencia o rama de la tecnología, que estudia el diseño y construcción de máquinas capaces de desempeñar tareas realizadas por el ser humano o que requieren del uso de inteligencia [4]. Estas máquinas son llamadas robots. Un robot es una máquina de manipulación automática reprogramable capaz de realizar diversos trabajos [5]. En la industria los robots posicionan y orientan materiales, piezas, herramientas o dispositivos especiales en las diferentes etapas de la producción industrial, a pesar de que no pueden parecerse a los seres humanos en la apariencia o llevar a cabo estas tareas de la misma manera (ya sea en una posición fija o en movimiento) [5].

Aunque los robots son diseñados para ejecutar, en general, algún tipo de movimiento, no se puede decir que estas máquinas tengan la capacidad de operar en un ambiente real de manera autónoma (aún requieren de la ayuda del ser humano para operarlos a ciertas distancias) [6].

La robótica ha logrado gran éxito en el mundo industrial con brazos robóticos, o manipuladores, ya que en una posición específica en la cadena de montaje, el brazo del robot puede moverse con gran velocidad y precisión para realizar tareas repetitivas como la soldadura por puntos, la pintura y armado de piezas [39].

1.1.1 Robótica Móvil

La robótica móvil es un área de la robótica que estudia aquellos robots que no tienen algún eslabón anclado a un medio físico fijo, como podría ser la tierra. Esta libertad les permite moverse en su entorno. Existen robots móviles acuáticos, voladores y terrestres. Todos ellos, por la manera en que están configurados y por la intrínseca relación que guardan con las disciplinas de la ingeniería mecánica e ingeniería electrónica, son considerados productos inherentemente mecatrónicos [39].

Los robots móviles pueden ser autónomos, lo que significa que son capaces de navegar un ambiente no controlado y sin la necesidad intervención humana. Como alternativa a la intervención humana, los robots móviles pueden confiar en los dispositivos de orientación que les permiten viajar una ruta de navegación pre-definido en un espacio relativamente controlada [38].

1.1.2. Robótica Colaborativa

El área que estudia la posibilidad que los robots colaboren entre sí para lograr un beneficio mutuo se conoce como robótica colaborativa [40]. La colaboración requiere que se tenga objetivos compartidos y exista interacción entre los robots. En la industria esta colaboración entre robots es muy usada en las cadenas de producción, donde cada robot tiene ciertas capacidades y requisitos previos antes de ejecutar cualquier tarea de fabricación. El Robot debe ser capaz de planificar de manera eficiente y ejecutar su camino en un entorno dinámico. Para que estos robots puedan interactuar con su ambiente y con otros, se suele usar un tipo procesamiento conocido como “inteligencia artificial”.

1.2 Inteligencia Artificial

Una definición de inteligencia en los seres humanos es la capacidad de recoger información de su entorno a través de sus sentidos [7], uniéndose al aprendizaje experimental y racionalizando para dar una solución a los problemas de la vida cotidiana [8].

Para que un robot tenga la autonomía que se requiere para desarrollar ciertas tareas o desempeñarse de acuerdo al escenario cumpliendo con los objetivos propuestos, este se le debe dotar de un tipo de inteligencia, esta inteligencia es conocida como inteligencia artificial. En 1956, durante la conferencia de Darthmounth, John McCarthy introdujo el

concepto de inteligencia artificial como la disciplina que se encarga de construir procesos, que al ser ejecutados sobre una arquitectura física basándose en la secuencia de entradas percibidas y en el conocimiento almacenado en tal arquitectura, producen acciones o resultados que maximizan una medida de rendimiento determinada [36].

La inteligencia artificial, que en un contexto general y según Kurzweill (1990), Shalkoff (1990) y Winston (1992), entre otros, se puede definir como el estudio y desarrollo de agentes racionales no vivos capaces de percibir, razonar y actuar emulando a los seres humanos [7]. En otras palabras, son agentes (robots) que perciben su entorno a través de información ingresada no por los sentidos como los seres humanos sino por sensores, pueden procesar dicha información y dar una respuesta programada, acción o salida a tal situación.

Entonces la inteligencia que se le programa a estos agentes no vivos, como tal no se define como inteligencia, sino como la racionalidad que se obtiene después de percibir su entorno (recibir entradas) a través de sensores, procesar dicha información y maximizar un resultado esperado. Este concepto se conoce como agente inteligente.

1.2.1 Agente Inteligente

Un agente inteligente se puede definir como un sistema computacional que es autónomo en las acciones que ejecuta en pro de la consecución de sus objetivos, que recoge información de su entorno a través de sensores y que tiene la capacidad de interactuar con otros sistemas semejantes.

Según Jeinnings y Wooldridge tres características básicas definen un Agente inteligente:

Proactividad: Se refiere a la capacidad de tomar la iniciativa para realizar las acciones necesarias para cumplir con los objetivos para los que son diseñados.

Capacidad de reacción: A partir de sus sensores, un agente debe estar en la capacidad de responder a los estímulos del ambiente con las respuestas adecuadas y en el tiempo adecuado, de forma activa.

Habilidades sociales: Un agente inteligente debe ser capaz de interactuar con otros agentes, inclusive humanos cuando es necesario para cumplir con sus objetivos. Las dos habilidades básicas son las habilidades de cooperación y de negociación [9].

1.2.2 Multi-Agentes

Los sistemas multiagentes definen una rama de las ciencias de la computación que ha venido evolucionando en las últimas décadas, rama que tiene como objetivo el estudio y desarrollo tecnológico para la solución de problemas a partir de la interacción del conjunto de robots autónomos o agentes inteligentes, que cooperan y se movilizan y que además poseen una arquitectura simulada y física avanzada. Los multi-agentes usan inteligencia artificial y se comunican para alcanzar un objetivo [12].

1.3 Simuladores en la robótica

Un simulador es un aparato o conjunto de aparatos que reproduce las funciones de otro, pero sin producir el efecto propio de él. Los simuladores juegan un papel importante en la investigación en robótica [32]. En comparación con la investigación en robots físicos, la simulación es más fácil de preparar, menos costosa, más rápida, más cómodo de usar y que permiten al usuario realizar experimentos sin el riesgo de dañar el robot.

El uso de simuladores como apoyo para la robótica no es algo nuevo [32]. Muchos de estos simuladores con el transcurrir el tiempo han ido evolucionando, haciendo la simulación cada vez más cercanas a la realidad. Estos ambientes simulados enfocan la investigación y desarrollo en técnicas de cooperación y aprendizaje, evitando los problemas típicos que se presentan en los robots físicos como son fallas en los componentes mecánicos, ruidos en la comunicación o en el reconocimiento de imágenes por parte de la cámara, facilitando la investigación y desarrollo de los agentes.

Los simuladores son tan útiles como su capacidad para reproducir las características más importantes del mundo real, liberan a sus desarrolladores de manejar los inconvenientes propios de los sistemas físicos, como son las características del hardware, la comunicación y el reconocimiento de imágenes. Para agilizar el desarrollo del software, los investigadores de la robótica, usan herramientas de simulación, de desarrollo propio o herramientas que se encuentran disponibles públicamente, lo que les permite tener una mejor aproximación de comportamiento del robot cuando este en el ambiente real [12].

1.3.1 Simuladores vs Realidad

Las diferencias entre el ambiente real y el simulado se deben a varios aspectos que se podrían separar como errores sistemáticos que son aquellos que se pueden estimar puesto que están siempre presentes en el sistema y los errores no sistemáticos que no son de fácil estimación ya que son de aparición aleatoria en el sistema [21].

Errores Sistemáticos: Desigualdad del diámetro de las ruedas, desalineación de las ruedas, potencias de los motores y las escalas de movimiento.

Errores no sistemáticos: Desplazamiento sobre suelo irregular, errores en deslizamiento de las ruedas debido a derrapes, aceleración, giros rápidos, mal contacto en el suelo, fricción, la diferencia de valores de peso [20].

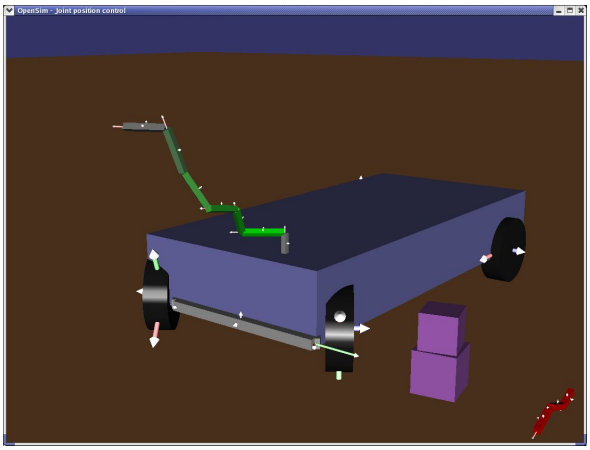
Surgen variaciones adicionales del comportamiento del robot simulado vs el robot real, ya que sin importar la calibración que se les dé a los robots en el ambiente simulado, en su transición al ambiente real, pueden cambiar en gran medida su comportamiento al esperado, siendo este tipo de problemas justamente la fuente de estudio y desarrollo tecnológico. Dentro de los simuladores objetos de estudio y más usados se encuentran:

1.3.2 OPENSIM

Su objetivo es proporcionar herramientas gratuitas y ampliamente accesibles para la realización de la investigación biomecánica y la ciencia. Es un simulador extensible de libre disposición y código abierto, que se enfatiza en el acercamiento realista de los robots móviles autónomos, permite a los usuarios desarrollar modelos de estructuras músculo esqueléticas y crear simulaciones dinámicas de movimiento. Generalmente, es usado para el modelado biomecánica, simulación y análisis de control del motor. Permite una amplia gama de estudios, entre esos la animación del movimiento humano y animal, tiene una comunidad de desarrolladores de software que contribuyen nuevas características. Es una de las aplicaciones emblemáticas de Simbios, un Centro NIH Biomédica Computación en la Universidad de Stanford. Fundada en 2004, conocido por proporcionar software líder y herramientas computacionales para el modelado y simulación de estructuras biológicas basado en la física. OpenSim fue diseñado para impulsar la investigación biomecánica, proporcionando un marco común para la investigación y un vehículo para el intercambio de modelos músculo esqueléticos complejos. posee un potente componente llamado IKOR para cinemática inversa pero

tiene la desventaja que se ha descontinuado el proyecto como lo explican es su página web [26], ver Figura 1-1.

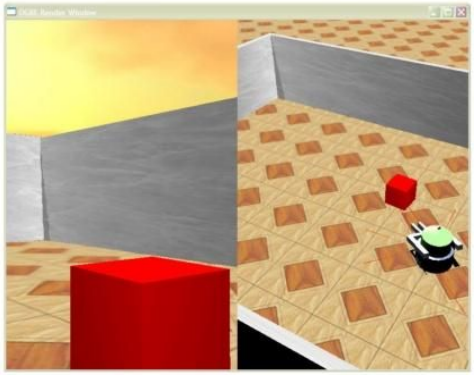
Figura 1-1: Simulador OpenSim [28].



1.3.3 XPERSim

Es un simulador en 3D basado en componentes de código abierto, que rápida y fácilmente construye una simulación precisa y realista de la morfología arbitraria de los robots y de los entornos en los que funcionan. XPERSim logra la visualización con alta calidad mediante el uso del motor de renderizado orientado a objetos gráficos 3D (Ogre) ver Figura 1-2, cuya dinámica es calculada con la librería ODE (Open Dynamics Engine) [23], simulando múltiples cámaras de manera simultánea de la escena a grandes velocidades.

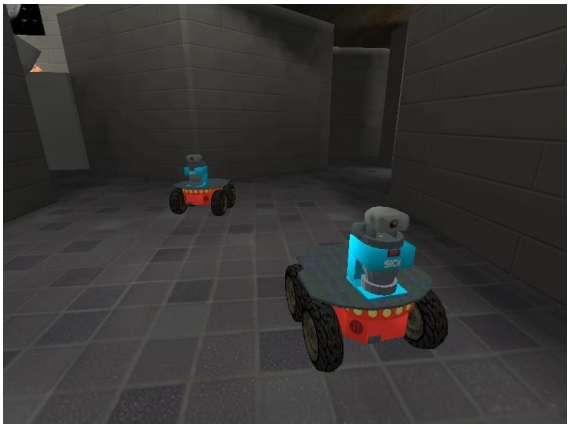
Figura 1-2: Simulador XPERSim. [23].



1.3.4 USAR Sim

(Sistema Unificado para la automatización y la simulación de robots) es una simulación de alta fidelidad de los robots y entornos basados en el motor de juego UnrealTournament [28]. Se pretende que sea una herramienta de propósito general de investigación con aplicaciones que van desde las interfaces de ordenador-humanos a la generación de comportamiento de grupos de robots heterogéneos. Además de las aplicaciones de investigación, USARSim es la base para el rescate de RoboCup competencia virtual robot (RoboCup), así como el IEEE fabricación virtual Competencia Automatización (VMAC), ver Figura 1-3.

Figura 1-3: Simulador USAR Sim [28].

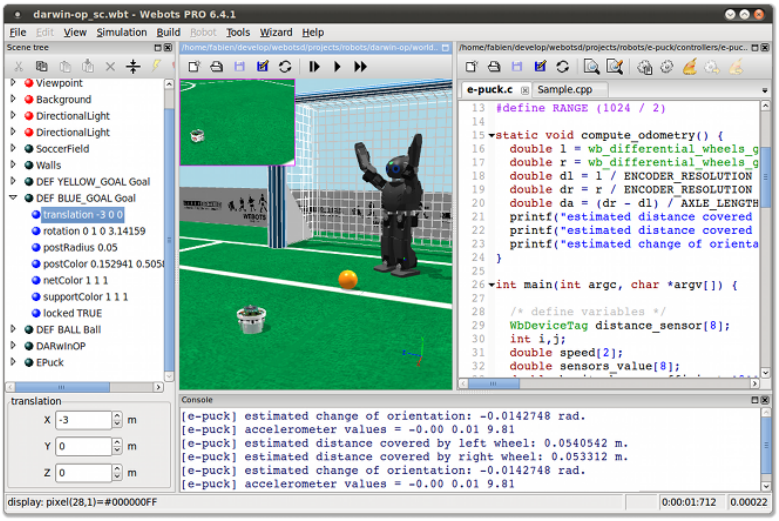


1.3.5 WebotsTM

Ha sido desarrollado en colaboración con el Instituto Federal Suizo de Tecnología en Lausanne. WebotsTM se trata de un producto comercial, básicamente es un robot a la medida donde el usuario tiene la facultad y los elementos necesarios para programar, actualizar y modificar la configuración completa de un robot móvil. WebotsTM permite compartir información y crear entornos personalizados y simularlos [32].

Esta tecnología trae muchos beneficios no solo a los usuarios sino a los mismos empresarios ya que genera el acceso a la investigación sin discriminación al alcance de todos los usuarios. Los beneficios que se tienen por adquirirla hacen que exista motivación para los desarrolladores ya que con la comercialización y la demanda de este tipo de sistemas encuentran los recursos para las investigaciones y el desarrollo, ver Figura 1-4.

Figura 1-4: Simulador WebotsTM [22].



2. Fútbol de robots

En este capítulo se trata el inicio y las generalidades del fútbol de robot como un sistema de control inteligente para multi-agentes y como escenario de exploración, estudio y solución tecnológica. En el fútbol de robots, la inteligencia artificial depende básicamente de los dispositivos de percepción que es el cómo obtiene el robot la información sobre el ambiente que lo rodea, esta percepción puede estar dada por los sensores del robot o por la interfaz con el servidor en el fútbol simulado y el tipo de procesamiento, que es cómo el robot elabora la información necesaria para actuar [8].

Existen unas características que deben cumplir los robots participantes en los torneos FIRA y RoboCup, que son los torneos más importantes a nivel mundial que fomentan el estudio y desarrollo de la tecnología a través del fútbol de robots. Cada categoría limita a los robots en cuanto a las dimensiones, movilidad, dispositivos de percepción y tipos de procesamiento.

2.1 Sistema Multi-agente en fútbol de robots

Si se define el fútbol de robots como un sistema multi-agente, cada agente tiene su propia función, debe analizar constantemente el entorno y a su vez deben trabajar en la consecución del objetivo común. Es por esto que es importante determinar y asignar roles a cada agente para reducir la complejidad a la hora de la toma de decisiones y se pueden excluir situaciones por improbabilidad durante el partido, sin que esto garantice que la decisión de un agente no entre en conflicto con las decisiones de los demás agentes [10]. Esta asignación de roles hace que el sistema sea sencillo, efectivo, rápido y robusto. Es necesario que los agentes tengan conocimiento sobre el comportamiento de otros agentes, de esa forma pueden usar tácticas fijas, predecir jugadas y accionarse de forma cooperativa, minimizando la comunicación explícita entre los jugadores. Aunque se debe tener en cuenta que este tipo de conocimiento sobre el comportamiento de sus

compañeros y oponentes, no siempre resulta efectiva, ya que no se puede tener toda la información sobre los diferentes entornos.

El fútbol de robot se puede ver como un sistema de control inteligente para multi-agentes, compuesto por dos o más robots móviles autónomos. Cada robot tiene su propio mecanismo de movimiento y por lo tanto se puede mover de forma independiente con otros robots [11].

Este juego está basado en el deporte humano llamado fútbol. Se desarrolla en un escenario tipo cancha donde los dos equipos competidores (formados por robots), deben desempeñar tareas específicas individuales y grupales que les permitan, a través del uso de inteligencia artificial, desenvolverse en un partido de fútbol, en tiempo real y cumpliendo el objetivo principal de encajar más veces que el equipo contrario el balón en la portería rival.

Para lograr que los robots tengan la autonomía necesaria, se les debe programar diferentes comportamientos para que puedan ser capaces de reaccionar adecuadamente en las situaciones que se les pueda presentar durante la competencia. Estos comportamientos se ponen en acción coordinadamente con otros robots de modo que se logra un plan de juego específico. Este plan de acción permite tomar decisiones de forma muy rápida con el objetivo de anotar más goles al equipo contrario [12]. A continuación se describirán las principales organizaciones que están trabajando en fútbol de robots.

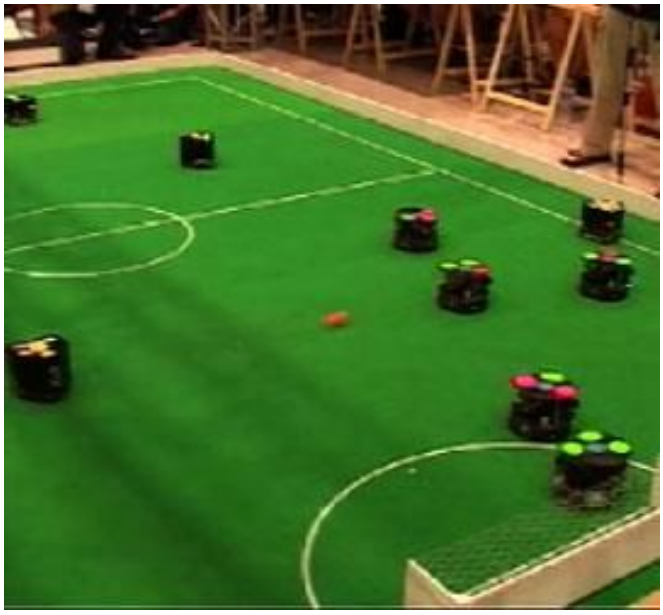
2.2 Organizaciones que promueven el fútbol de robots

2.2.1 Robocup

La Robocup es la asociación internacional que fomenta la investigación en Inteligencia Artificial, la colaboración multi-agente y los torneos de fútbol de robots. El objetivo oficial de la RoboCup es:
“A mediados de siglo 21, un equipo de robots humanoides ganará el partido de fútbol contra el ganador de la Copa del Mundo.” Este es un hito similar al conseguido por el programa AlphaGo de la compañía Google DeepMind que venció al campeón coreano de Go, Lee Sedol en Marzo del 2016 o el logrado por Deep Blue de IBM derrotó al campeón Garry Kasparov en 1997. La Robocup se centra en una serie de concursos donde los robots en diferentes ligas juegan fútbol unos contra otros. Fue establecido con el ánimo

de promover la investigación en el campo tecnológico y de robótica inteligente, como resultado de la creación del primer torneo de copa mundial de fútbol de robots. En esta competencia se evidencia un problema estándar, donde se encuentra una amplia gama de tecnologías integradas y examinadas. Los equipos participantes de todo el mundo, además de tener la oportunidad de mostrar sus innovaciones, inventos y propuestas a nivel de hardware y de software en sus equipos de robots, tienen la oportunidad de medirse con las mejores estrategias, es un evento que promueve la cooperación y el código abierto donde se reciben a muchos investigadores y creativos, que desean participar en estos proyectos con grandes aportes. Con la realización de estos torneos los diferentes equipos se esfuerzan porque sus robots tengan las actualizaciones y la tecnologías necesarias a nivel de software como de hardware para poder realizar una buena presentación. Es en estos torneos donde se intercambian conocimientos, ideas innovadoras y soluciones a circunstancias no previstas, todo con el ánimo de que cada equipo esté en la capacidad de mejorar en la competencia frente a los demás, ver Figura 2-1 [13].

Figura 2-1: Robocup :Soccer Small Size League [13].



2.2.2 FIRA

FIRA (Federación Internacional de Fútbol Robótico Asociado) es una competencia de alto nivel técnico-científico, desarrollado por el profesor Jong-Hwan Kim, de KAIST (Instituto Coreano de Tecnología) en octubre de 1995 como un campo de pruebas de efectos múltiples para el aprendizaje y la aplicación en el campo de alta tecnología, tales

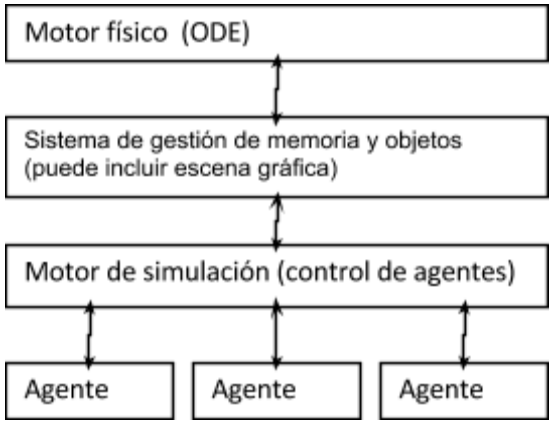
como el análisis de imágenes, la inteligencia artificial, sensores, la comunicación, el control electrónico de precisión, motores de accionamiento, así como software y hardware. Desde entonces ha crecido constantemente a medida que los científicos cada vez más jóvenes participan. La primera competencia internacional de robótica mundial de fútbol, se celebró en KAIST en noviembre de 1996, y el segundo se jugó en junio de 1997 en el mismo lugar. Sólo después de tres años desde el nacimiento de fútbol de robots el 5 de junio de 1997, fue fundada la federación de fútbol internacional, compuesto por 34 países (FIRA: Federación Internacional de Robot-Soccer Association). Después de la inauguración oficial de la Fira se llamó "FIRA Robot World Cup" [14], ver Figura 2-2.

Figura 2-2: Micro Robot World Cup Soccer Tournament (MiroSot) [14].

2.3 Simuladores usados en competencia

Los componentes principales de un simulador usados en las competencias de fútbol de robots y que se ilustran en la **Figura 2-3**, son el motor de simulación, el sistema de gestión de memoria y objetos, y el motor físico.

Figura 2-3: Control de flujo y el flujo de datos entre los componentes principales del simulador. (Adaptado de [12]).



Entre las características principales con las cuales debe contar el simulador de fútbol de robots está **el motor físico**, que es responsable de la actualización, ubicaciones y velocidades de los objetos simulados de acuerdo con un modelo físico, realizando detección de colisiones el cual debe representar la realidad tanto como le sea posible. Una de las opciones más usadas es la utilización de Open Dynamics Engine™ (ODE) que es una biblioteca gratuita y de calidad industrial para la simulación de dinámicas de cuerpos rígidos articulados [17]. ODE Es rápido, flexible y robusto, y tiene funciones de detección de colisión. ODE está siendo desarrollado en su mayoría en C++ por Russell Smith con la ayuda de varios colaboradores. Esta librería tiene sus variantes para cada lenguaje, en el caso de java se tiene ode4j.org.

El componente de gestión de memoria y objetos se encarga de armar la escena en la cual habitan los agentes para el cálculo de la información sensorial de cada uno y debe controlar los procesos y la interacción que se maneja entre ellos [12]. Este componente posee recursos limitados en cuanto a tiempo, cantidad de memoria y procesador disponible, gran parte de ello se debe a que se trata de un problema en tiempo real.

El motor gráfico o de *render* es el que proporciona funciones de renderizado 2D o 3D para poder visualizar la simulación. Se suele manejar librerías/APIs gráficas como Irrlicht, OpenGL,OGRE o DirectX [17], ver Figura 2-4 .

Figura 2-4: Algunos Motores gráficos frecuentes usados en simulación.

Irrlicht Engine	OpenGL
OGRE	Microsoft DirectX

El motor de la simulación controla los procesos, el tiempo y gestiona la comunicación. Contiene el ciclo principal del simulador que permite un bucle simple y directo, está pendiente de las acciones enviadas por los agentes tan pronto como llegan, diseñado para controlar los eventos de las simulaciones. Debe funcionar en múltiples hilos para permitir el manejo paralelo de los agentes [19].

2.3.1 SimuroSot Simulador de la FIRA:

Consiste en un servidor que cuenta con los entornos de juego de fútbol (zona de juegos, robots, tablero de puntuación, etc.), y dos programas clientes con las estrategias de juego. Las estrategias se escriben en C++ o en Lingo y en una pantalla gráfica a color en 3D muestra el partido. Los equipos pueden hacer sus propias estrategias. La plataforma de simulación en 3D de 5 vs 5 y 11 frente a 11 juegos están disponibles en el sitio web de FIRA. Escrito en Macromedia Studio 8.5. La detección de colisiones y dinámica de los modelos son proporcionados por el motor físico Havok, incluido en el Director 8.5.

Permite el desarrollo de estrategias en tiempo real, desarrollar acciones, algoritmos y estrategias en el fútbol de robots, provee un ambiente de entrenamiento y aprendizaje de estrategias para cada robot, puede utilizarse para medir la viabilidad y avances en la estrategia de juego de cada equipo. SimuroSot permite simular el campo de juego, los robots y la pelota, el uso de la cinemática y dinámica para modelar los movimientos de los robots y la pelota. El simulador ha sido compilado usando Visual C++ 6.0 y puede ser corrido bajo Windows 98 o una versión superior, ver Figura 2-5 [27].

Figura 2-5: SimuroSot Simulador de la FIRA [27].

2.3.2 Simulador SimSpark

Es un sistema de simulación multi-agente en entornos tridimensionales creado como iniciativa de fomentar el estudio y la investigación en la robótica inteligente, siendo este justamente un problema estándar que abarca un amplio campo en la tecnología. Su objetivo es proporcionar un alto grado de flexibilidad para la creación de nuevos tipos de simulaciones. La versión 3D del simulador de Robocup, es una versión mejorada, solucionando en gran parte los problemas que se presentan de la simulación al mundo real, originados por la simplificación realizada por el simulador de muchos aspectos reales físicos. En comparación con simuladores especializados, los usuarios pueden crear nuevas simulaciones mediante el uso de la descripción de una escena. SimSpark fué utilizado con éxito para la primera competición oficial en la RoboCup en su liga de Simulación 3D. La simulación de fútbol para este torneo se desarrolló en paralelo con el simulador SimSpark. En la versión inicial sus jugadores fueron modelados como esferas físicas en un mundo tridimensional. Desde entonces SimSpark creció considerablemente y ahora es compatible con reproductores de humanoides con los cuerpos articulados [22], ver Figura 2-4.

Figura 2-6: Simulador SimSpark RoboCup. [19].

2.3.3 GrSim

Es un simulador 3D moderno multi-robot desarrollado principalmente para la simulación de la Liga de Fútbol para robots pequeños de la RoboCup (RoboCup Small Size League Soccer). El software es útil para las personas que participan con pequeños robots en proyectos de fútbol, así como investigadores en el campo de los sistemas multi-agente. GrSim es un software libre, publicado bajo los términos de la Licencia pública general GNU (GNU General Public License), ver Figura 2-7 [28].

Figura 2-7: Simulador GrSim [25].

2.3.4 NEWNEU 3D

Es un simulador de fútbol de robots, que consiste en un sistema de alta fidelidad de la realidad con visualización en tiempo real, permitiendo el desarrollo de estrategias, para la toma de decisiones en la interacción humano-computadora. Esta plataforma simula casi todos los procesos en una competencia de fútbol de robots reales, con efectos 3D de colisión.

El desarrollo de un simulador con tal calidad de definición de la realidad promueve la percepción del usuario a la realidad dentro del juego, también permite prever ciertas inconsistencias en el desarrollo de las estrategias o incluso a nivel de software o hardware usado en el juego de fútbol de multi-robots [28].

2.3.5 UCHILSIM

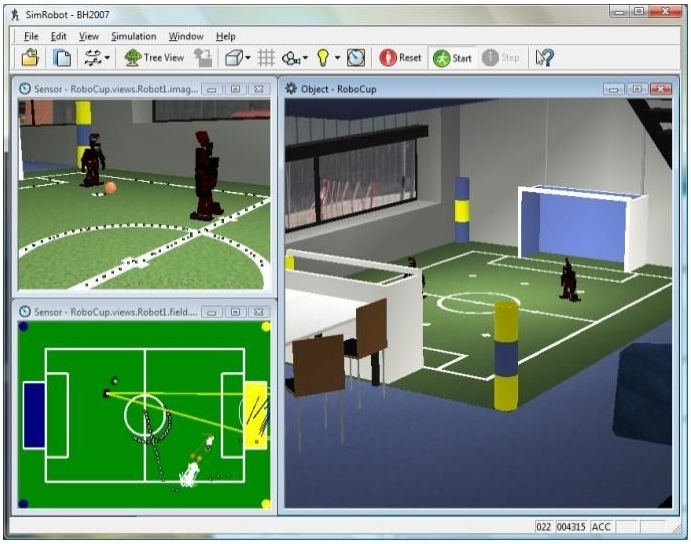
Es un simulador robótico destinado principalmente a la liga de RoboCup de cuatro patas. Se reproduce con exactitud la dinámica de los movimientos robot AIBO de SONY y sus interacciones con otros objetos en el campo de fútbol. Las representaciones gráficas dentro del campo también poseen un alto nivel de detalle. Un objetivo principal del simulador es convertirse en una plataforma para el aprendizaje de comportamientos complejos robóticos que pueden ser directamente transferidos a un medio ambiente real. El logro de una simulación realista de un AIBO robot es una tarea particularmente difícil debido a la gran variedad de sensores, actuadores y su precio reducido, este sistema se considera como una de las mejores plataformas robóticas que jamás se ha hecho [30]. (ver figura 2-8.)

Figura 2-8: Simulador UCHILSIM [30].

2.3.6 Sim Robot

Este proyecto está basado en OpenGL y ODE. Posee una variedad de robots con patas y ruedas, ha sido simulado con éxito por SimRobot especialmente los de la RoboCup que es un simulador más entre muchos otros simuladores de RoboCup, SimRobot, por supuesto, tiene muchas similitudes con otras aplicaciones, así como algunas características que aún no han sido ejecutadas por otros. Tiene grandes ventajas en cuanto a la simulación de la lectura de los sensores como la cámara, y ejecuta órdenes dadas por el controlador o el usuario. Incluso la mayor parte de la visualización se integran en el núcleo. La interfaz es amigable y se puede acomodar los objetos en la escena [28], ver Figura 2-9.

Figura 2-9: Simulador SimRobot [31].



La **Tabla 2-1** describe los simuladores más comunes en fútbol de robots por sistemas operativo, Lenguaje usado y Licencia. Debido que queremos tener mejor control del sistema optamos por crear uno propio Furo Java el cual detallaremos en el siguiente capítulo.

Tabla 2-1: Comparación de los simuladores

Simulador	Sistema Operativo	lenguaje	licencia
SimuroSot	Win	Macromedia Studio	No comercial
SimSpark	Win / Linux /Mac	C++,Ruby	GPL
XPERSim	Win / Linux /Mac	C++, JAVA	GNU
GrSim	LINUX	C++	GNU
NEWNEU	Windows		
USARSim	Win / Linux /Mac	C++,JAVA	GNU
OPENSIM	Win / Linux /Mac	C#	GNU
UCHILSIM	WINDOWS	C++	GNU
Sim robot	Win / Linux /Mac		GNU
WEBOTS	Win / Linux /Mac	C,C++, Java,	Propietaria

3. Simulador de fútbol de robots FuroSimSpark

3.1 Antecedentes

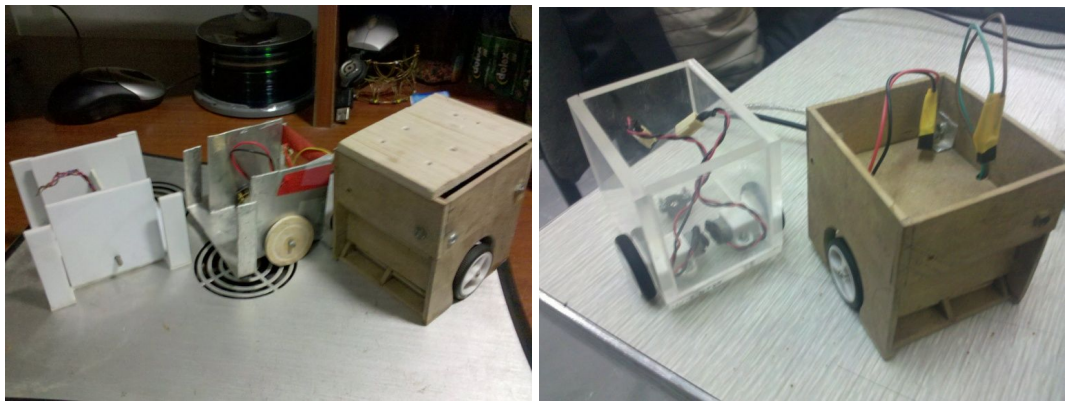
3.1.1 Implementación robot físicos

La implementación del robot físico FURO está compuesto por tres partes: chasis del Robot, tarjeta electrónica y software de control.

Chasis del robot

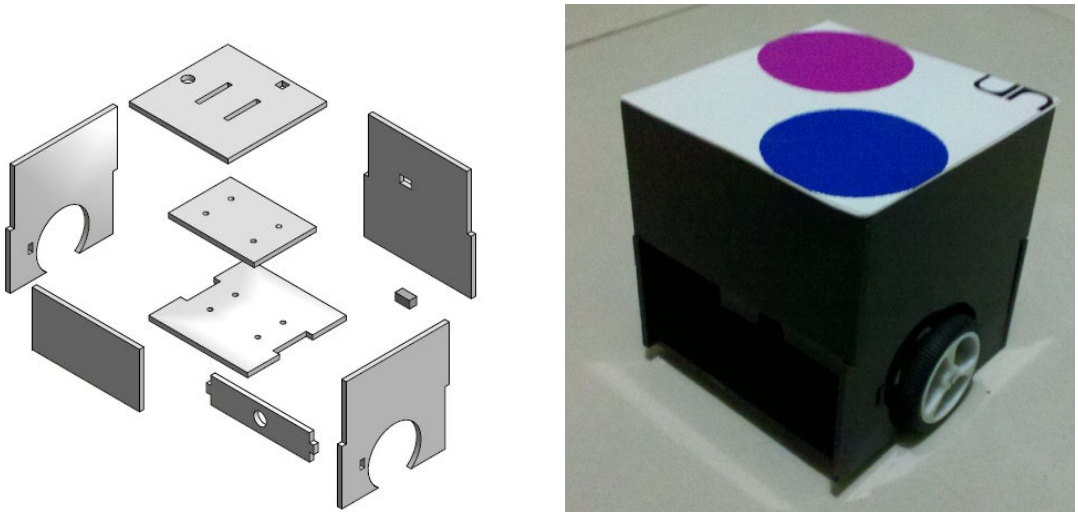
La construcción de la carcasa para el robot móvil se modelo con dos ruedas diferenciales que se controlan independientemente por dos motores que le permiten movimientos en línea recta, giros o arcos sobre su propio eje. Este diseño ha pasado por varios cambios iniciando en MDF y aluminio y terminaron en acrílico [37]. Estos modelos tienen gran complejidad en su proceso de armado y existía más probabilidades que el robot tuviera un comportamiento final no deseado, ver Figura 3-1.

Figura 3-1: Chasis Robot Furo [37].



Para la versión final del robot Furo, se modeló en Autocad, cada una de las partes y fué construida usando corte láser en acrílico (por las facilidades que daba el armado y los mejores acabados), ver Figura 3-2.

Figura 3-2: Robot Furo versión acrilico [37].



Tarjeta electrónica

El circuito electrónico está compuesto por cuatro módulos: fuente de potencia, microcontrolador, comunicación de radio y driver de motores.

La fuente de potencia del circuito está basada en el regulador LM3478, ver figura **3-3**

Figura 3-3: Fuente de Potencia.

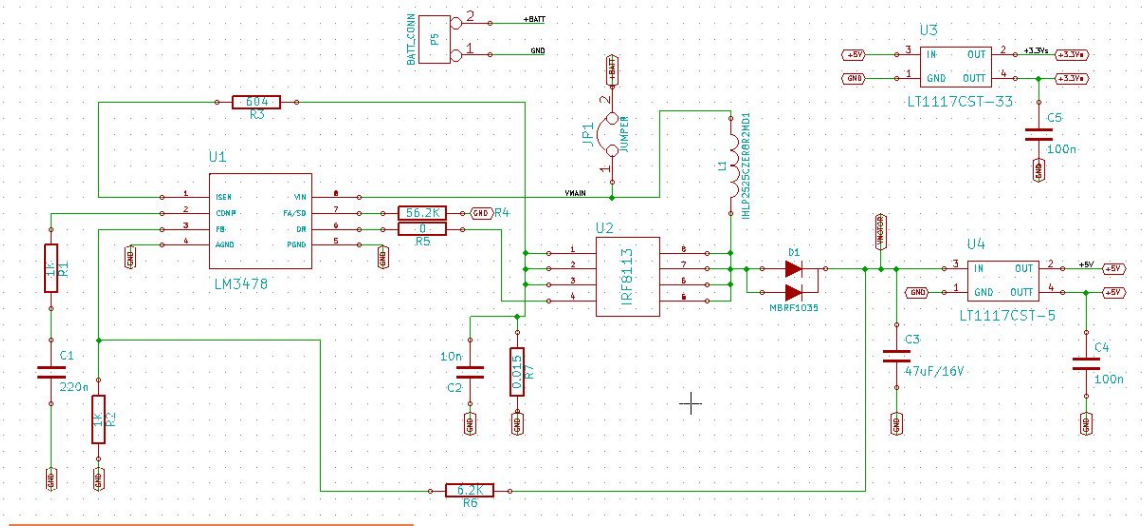


Figura 3-4: Circuito Microcontrolador ATMEGA168V.

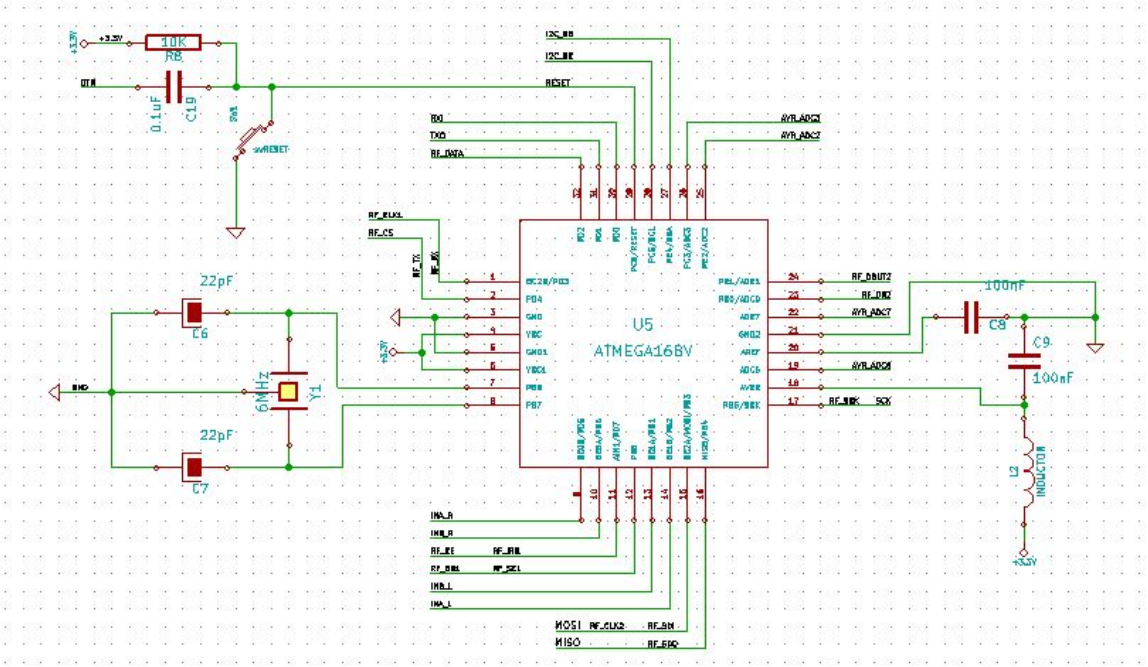


Figura 3-5: Módulo de Radiofrecuencia.

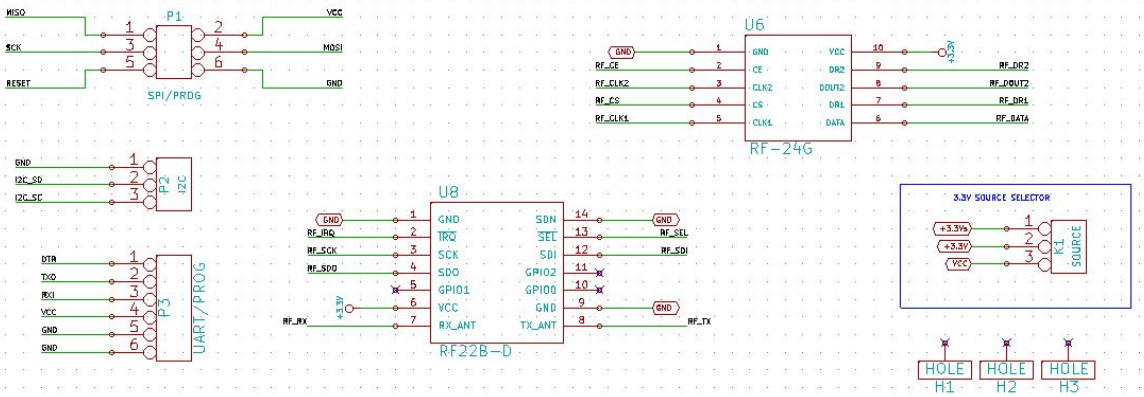


Figura 3-6: Controlador puente H MPC17529 Motor

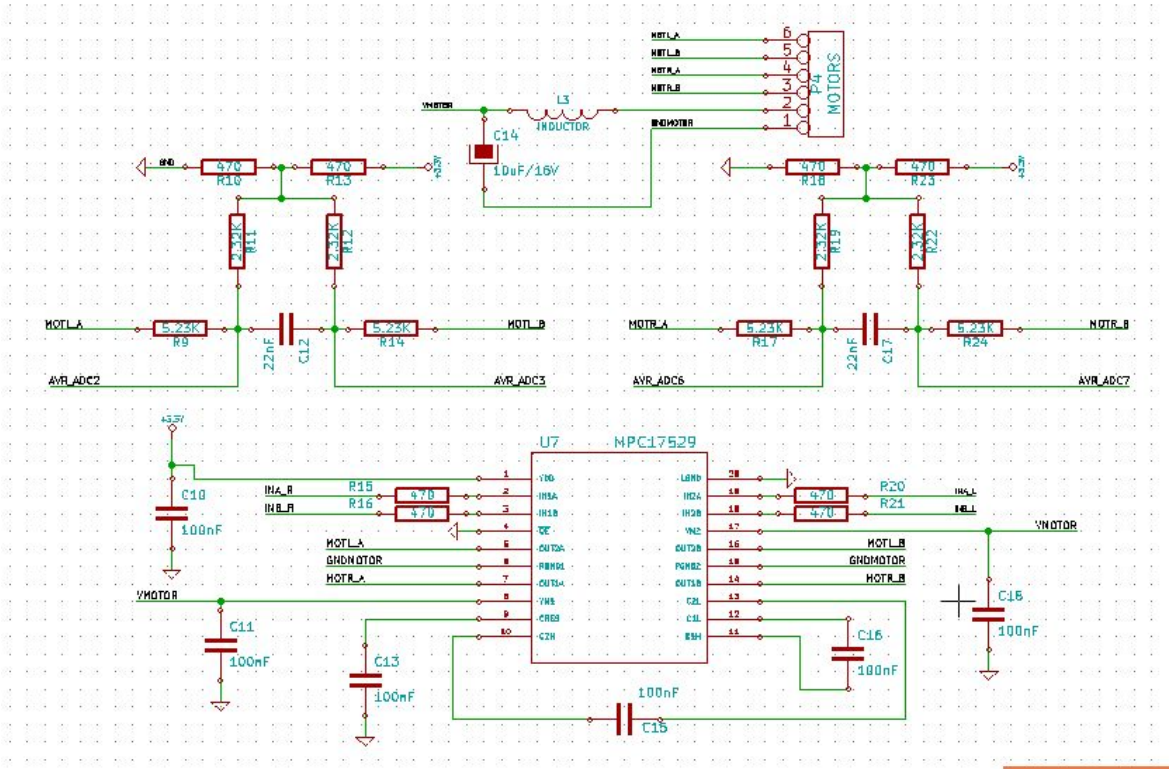


Figura 3-7: Tarjeta UN FuRo, modelo 3D

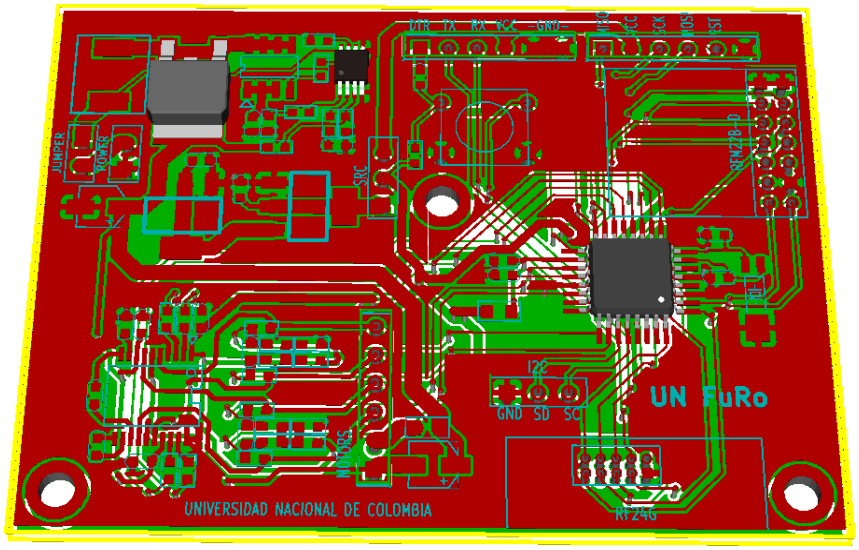
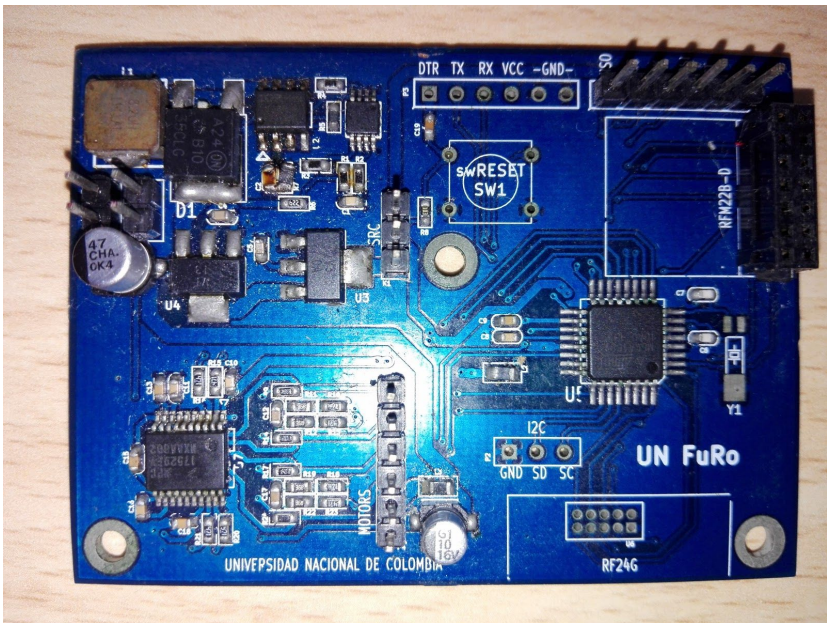


Figura 3-8: Tarjeta UN FuRo, placas soldadas



Software de control Atmel Studio

Para el desarrollo del software de control de la tarjeta se utilizó la plataforma de desarrollo de Atmel Studio 7. Este entorno de desarrollo proporciona una interfaz sencilla para el generación y depuración del código de los microcontroladores Atmel®. La aplicación que se desarrolla para esta tarjeta tiene dos modos. El primero de ellos el de la tarjeta receptora del robot encargada de recibir las instrucciones que debe realizar cada Robot. Esta tarjeta puede recibir alguno de los siguientes comandos:

Tabla 3-1: Instrucciones tarjeta receptora

Tecla	Instrucción
‘S’	detiene el PWM
‘F’	Coloca los dos PWM hacia adelante
‘L’	Gira el robot a la Izquierda
‘B’	Coloca los dos PWM hacia atrás
‘R’	Gira el robot a la derecha

El segundo modo es como tarjeta transmisora desde el computador personal su objetivo es recibir por medio del puerto Serial del computador las instrucciones que se tienen que ejecutar y las transmite vía Radio frecuencia a las tarjetas de los robot. Usa la instrucción *commtrans()* para realizar esta comunicación con el computador. Este puede recibir alguno de los siguientes comandos:

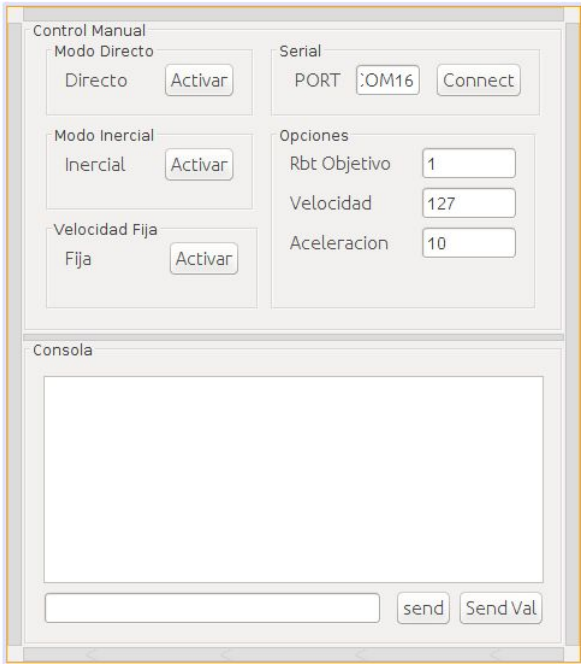
Tabla 3-2: Instrucciones tarjeta transmisora

Tecla	Instrucción	Robot
‘s’	detiene todos los robots	todos
‘t’,’i’,’z’,’b’	envía una F al robot correspondiente	h,l,g,j
‘f’,’j’,’x’,’n’	envía una R al robot correspondiente	h,l,g,j
‘g’,’m’,’c’,’q’	envía una B al robot correspondiente	h,l,g,j
‘h’,’l’,’v’,’e’	envía una L al robot correspondiente	h,l,g,j

Programa Furo serial

Para realizar las primeras pruebas de la tarjeta se desarrolló un aplicativo en Java para el manejo del robot por medio de un control de videojuegos los detalles de la la aplicación pueden consultarse en el anexo 2.

Figura 3-9: Programa Furo serial



Torneo de fútbol de robots Controlados por Joystick

Figura 3-10: Afiche del Torneo

Para realizar las pruebas de los robots se realizó un torneo interno de fútbol de robots el cual tenía como objetivo principal realizar las pruebas de rendimiento de los robots en esta competencia. Se tuvo un pequeño patrocinio por parte de una donación google muñeco android.

Figura 3-11: Equipo concursante

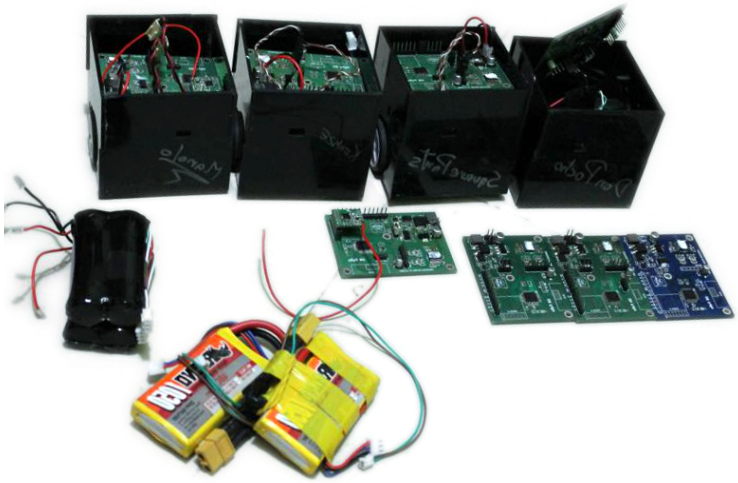
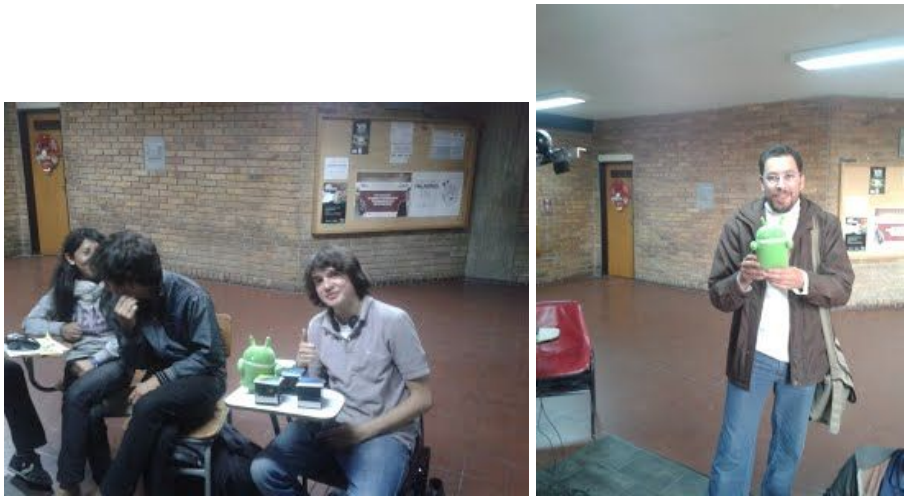


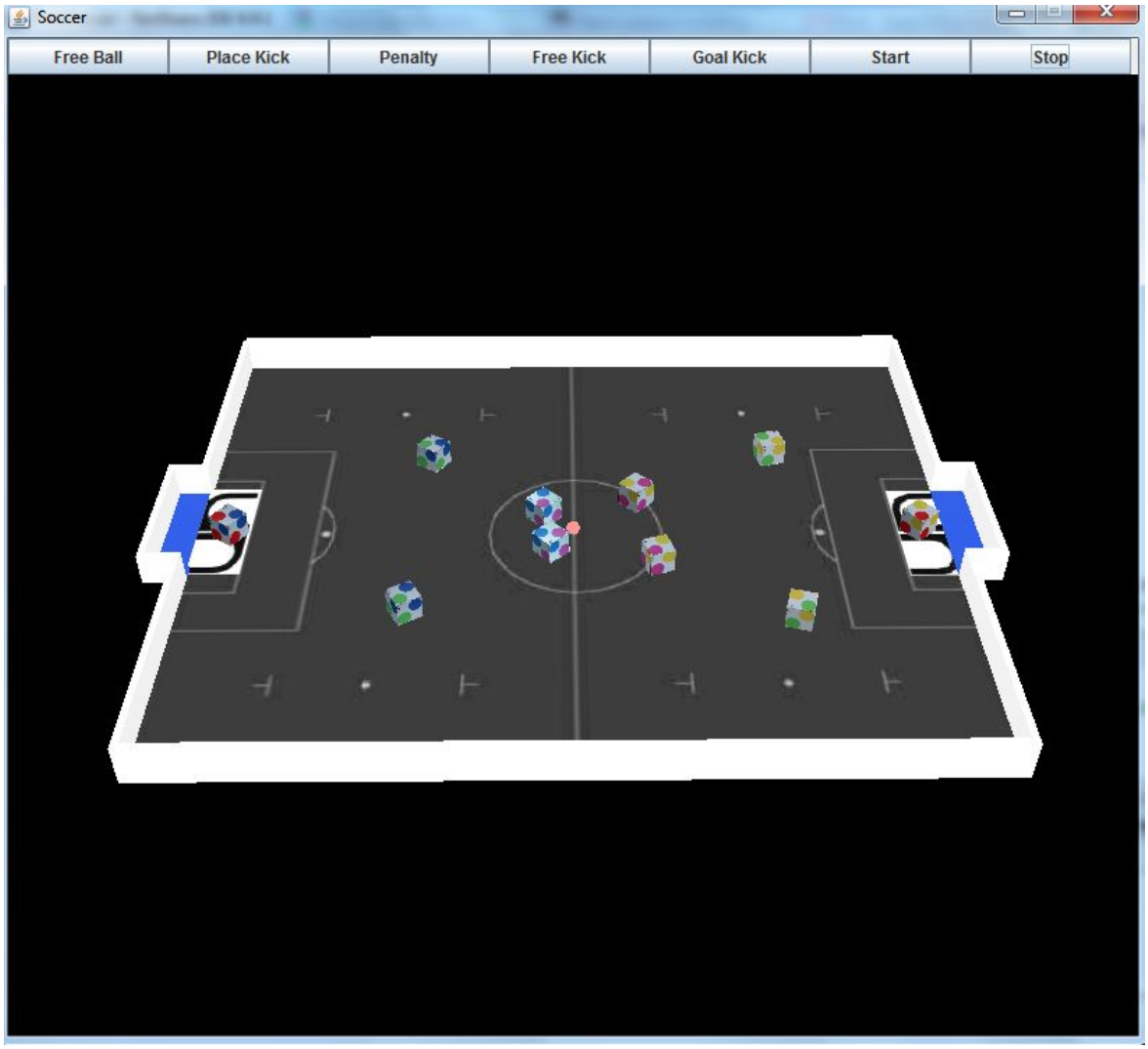
Figura 3-12: Ganadores del torneo.



El aprendizaje que dejó este evento es que para una persona común es muy complicado controlar un robot de movimiento diferencial y los robots tienen mucha dificultad si el suelo no es totalmente regular. Al final de todo se dejaron las memorias alojadas en https://sites.google.com/site/unfuroalife/

3.1.2 Simulador de fútbol de robots Furo JAVA

Figura 3-13: Vista de la versión de Furo 3D.



La motivación que llevó a no utilizar un simulador ya realizado y desarrollar las estrategias en otros lenguajes como C++ o Java, es el interés de tener una herramienta propia para implementarla como se considere conveniente, y sobre todo la necesidad de implementar unas librerías y estrategias realizadas dentro de la Universidad Nacional de Colombia [34].

Furo empezó como una idea para participar en el torneo mundial de fútbol de robots organizado por la FIRA. Inicialmente, se deseaba participar en la categoría MiroSot, pero a medida que se desarrollaba el proyecto fueron evidentes las diferencias de tecnologías y conocimientos con otros países, así que la decisión fue continuar progresivamente con el desarrollo de los agentes y en paralelo la creación de un simulador, basándonos en el construído por la FIRA para la categoría simurosot. Esto permitió crear nuestras propias herramientas, proveer un ambiente para nuevos desarrollos, estudiar y entender tecnologías desarrolladas alrededor del mundo con la idea de que en un futuro cercano, Colombia sea parte de las competencias internacionales [34].

3.2 Software de Simulación FuroSimSpark

A continuación se define de manera específica la creación, funcionalidad e implementación del simulador para sistema multi-agentes FuroSimSpark. Así como los aportes para el fútbol de robots y la inteligencia artificial.
El equipo de fútbol robótico de ALife, está conformado por robots con ruedas diferenciales, robots con las características que aplican en la liga tamaño pequeño (Small Size), de cinco jugadores de no más de 7 cm de arista y en la que la pelota es de color naranja del tamaño de una pelota de golf, ver figura 3-14 [14].

Figura 3-14: Liga Small Size [14]

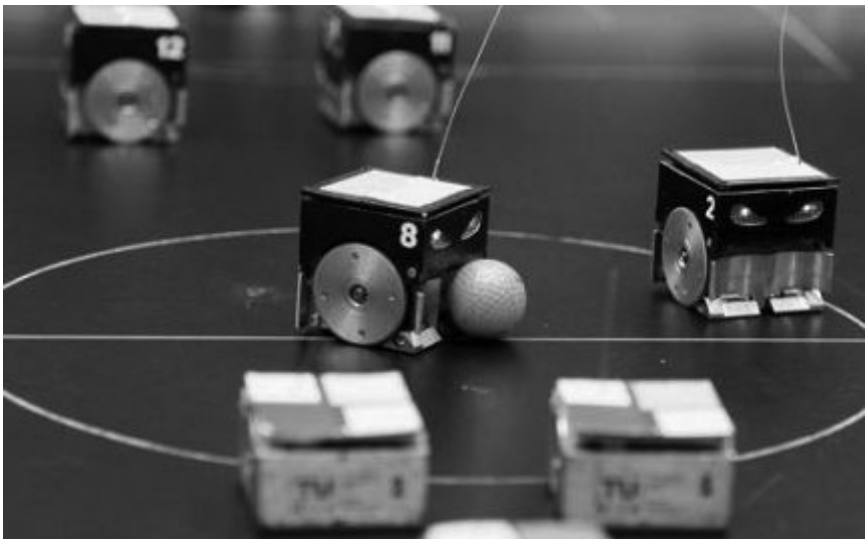


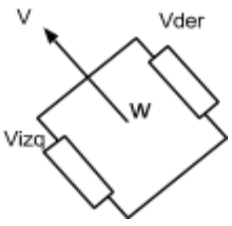
Figura 3-15: Vista de la versión de FuroSimSpark.

La categoría Small Size está regida por el reglamento de la FIRA (www.fira.net), donde existe un programa simulador de partido de fútbol, en donde se simulan los aspectos más importantes del mundo real, como la pelota, los robots (jugadores de fútbol) y el campo de juego.

Para el cálculo de la velocidad y la Aceleración los simuladores básicos manejan la siguiente fórmula, ver Figura 3-16.

Figura 3-16: Caso típico de cálculo de movimiento del robot. [18].

$$A = \frac{V_{der} - V_{izq}}{2r} \qquad V = \frac{V_{der} + V_{izq}}{2}$$



El simulador FuroSimSpark, es un sistema multi-agente genérico y flexible desarrollado para el equipo de fútbol robótico del grupo de investigación ALife, es una versión modificada de SimSpark que es el software de simulación oficial de la Robocup 3D. Como el SimSpark, el Simulador FuroSimSpark está compuesto por tres partes que son:

1. El servidor: El servidor es el que gestiona la simulación, siendo el responsable de modificar el estado el estado de la simulación en un ciclo continuo. Los objetos en escena cambian su estado de velocidad, posición o velocidad angular, debido a los demás objetos con los que interactúan, a la gravedad y a la fricción. Otra responsabilidad del servidor es realizar seguimiento a los procesos de los agentes conectados.

2. El monitor y registro de jugadas: este se conecta con el servidor a través de un sockets, por el cual recibe continuamente los datos actualizados de la simulación ya sean completos o incrementales con respecto al estado anterior para poderlos visualizar de forma gráfica permitiendo visualizar el desarrollo del juego.

3. Los agentes: Se desarrolló un agente que muestra la forma simple de interactuar con el servidor para leer las percepciones y utilizar los efectores, de esta manera personifica los comportamientos en el simulador.

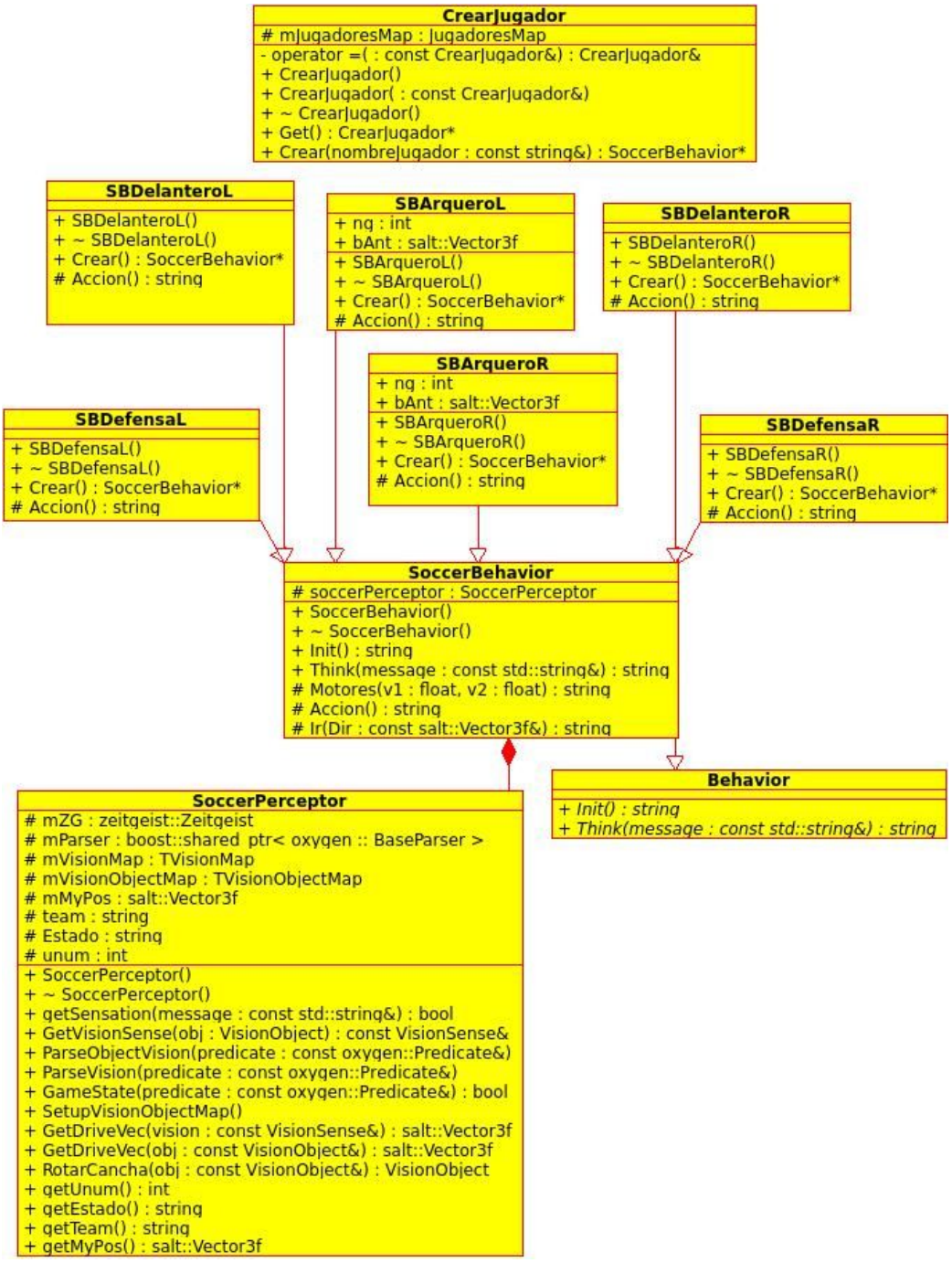
3.2.1 Requerimientos para desarrollar el proyecto

Para el desarrollo de todo el proyecto se requirió interactuar y aprender sobre diversas áreas de investigación y desarrollo, las cuales se encargan de la construcción de la estructura mecánica de los robots, su arquitectura de hardware y controladores, desarrollo del software en áreas tales como la imagen, análisis, el procesamiento de la información recogida por los sensores. Otra parte del grupo debe centrar su esfuerzo en el desarrollo de una estación base eficiente también llamada “simulador” de enfoque cooperativo entre los agentes. Esta última es el objeto de desarrollo del presente trabajo y requiere de lo siguiente:

1. Software libre: SimSpark,
2. Lenguaje de programación: C++
3. PC con requerimientos mínimos: 1.0 GHz, 128 tarjeta de video, 256 Mb en RAM, Disco duro de 20Gb.
4. Sistema Operativo: Linux.

Para la construcción de la aplicación se describe la estructura del sistema mostrando la sus componentes por medio de un diagrama de clases, ver Figura 3-2.

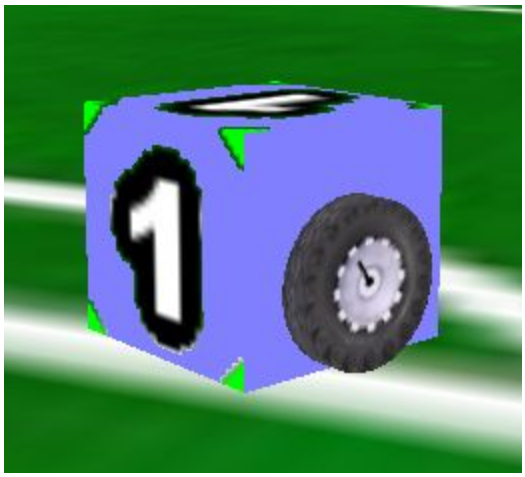
Figura 3-17: Diagrama de clases programa agente FuroSimspark.



Para el desarrollo del simulador FuroSimSpark, se modelaron los agentes físicos y sus cualidades como son medidas, peso y velocidades. De esta forma las estrategias creadas de forma virtual se puedan asemejar lo más posible al entorno real. Con este propósito se usó el lenguaje de programación Common Lisp [30], que utiliza una estructura de datos que incluye vectores, strings y listas.

Al igual que el agente físico, el agente simulado que se modeló está compuesto de un chasis en forma cúbica con arista de 7 cm de lado, a los costados dos ruedas de 4 cm de diámetro y 2 cm de ancho unidos a dos motores conocidos en el simulador como efectores, capaces de dar movimiento diferencial al robot a partir de la diferencia de velocidad de cada uno. Adicional a esto, el robot simulado cuenta con perceptores como acelerómetro y giroscopio que le permiten tener una ubicación y ángulo del robot y otros que permiten recibir los mensajes de los otros robots, ver Figura 3-18.

Figura 3-18: Modelo agente FuroSimspark.



El objetivo del simulador FuroSimSpark es que los agentes virtuales del equipo de ALife puedan jugar fútbol y meter goles en la portería contraria. Para esto, los agentes de manera individual deben llevar a cabo una serie de pasos.

1. Encontrarse en posición vertical y ubicar su posición dentro del campo de juego.
2. El agente debe encontrar la pelota y dirigirse hacia ella.
3. Una vez que tiene enfrente a la pelota, debe ubicar y trasladarse con la pelota hacia la portería contraria.
4. Al tener la portería cerca, el agente debe empujar la pelota hacia la portería. Si en la ejecución de alguno de los pasos anteriores, el agente llega a perder el la pelota debe volver al primer paso.

3.2.2 La comunicación entre los agentes y el servidor

La comunicación entre los agentes y el servidor se realiza a través de mensajes utilizando el protocolo TCP, una vez los robots son conectados deben enviar los mensajes de inicialización. Para la transferencia de datos entre el servidor y los agentes se empaquetan en expresiones simbólicas usando el lenguaje Common Lisp [30], el agente interactúa con el servidor comunicándose con sensores y efectores como lo haría en el mundo real. Del mismo modo el agente envía al servidor los comandos de los efectores. Los detalles sobre los formatos y los contenidos de los mensajes se dan a continuación:

Mensajes de efector

Efector de creación: Luego de que un agente es conectado al servidor debe enviar un mensaje de creación para cargar la representación física del modelo, de la siguiente forma **(scene <ubicacionDelArchivo>)** e indicarle la ruta del archivo. Para el simulador FuroSimSpark el ejemplo **(scene /rsg/agent/newfuro.rsg)**

Efector de inicialización: Luego que el agente sea creado se le asigna un número y un equipo con el efector de inicio **(init (unum <numeroDeJugador>) (teamname <nombredelequipo>)).** Ejemplo de este **(init (unum 2) (teamname FuroTeam)).**

Efector de Ubicación: Este permite ubicar el agente en el campo de juego y el ángulo de orientación del mismo **(beam <x> <y> <rot>).** Ejemplo de este **(beam 10.0 5.0 0).**

Efector de Giro: El agente puede controlar cada uno de los efectores por medio de su identificador y su velocidad angular en radianes por segundo. Este efector se usa para girar de la siguiente forma: **(<efector> <velocidad>)**. Un ejemplo de esto puede ser **(motorder 100)**. Al recibir esta información el servidor girará el motor derecho 100 radianes por segundo en sentido horario y continuará con este movimiento hasta recibir una nueva orden sobre este motor.

Efector de mensaje: La comunicación entre los agentes se realiza a través de mensajes de radiodifusión con el siguiente formato **(say <mensaje>),** donde el mensaje es un texto con máximo 20 caracteres en formato ascii de impresión con valores entre 33 y 126 sin el 40 y 41 que son los paréntesis “()”. Un ejemplo de esto puede ser **(say holaMundo)**. Solo los agentes que posean un perceptor de escucha pueden recibir el mensaje enviado.

Mensajes de perceptor

Perceptor articulación de bisagra: Recibe la información del nombre del perceptor y el ángulo de posición del eje en el cual se encuentra. EL formato descrito es **(HJ (n <nombre>) (ax <ax>)).** Ejemplo de este **(HJ (n EncoderDer) (ax -1,02)).**

Perceptor de escucha: Los agentes no tienen permitido comunicarse entre sí directamente pero pueden intercambiar mensajes a través del servidor. Es por esto que este perceptor permite recibir mensajes que otros jugadores enviaron. El siguiente es el formato como se visualiza **(hear <tiempo> self/<dirección> <mensaje>).** Ejemplo de este **(hear 5.8 self voyalataque)** o **(hear 5.8 -12.7 pasomododefensa).**

Perceptor de estado: Entrega información relevante al estado interno del agente más exactamente sobre la batería y la temperatura actual del agente. El formato de este perceptor es **(AgentState (temp <grados>) (battery <Porcentaje>)).** Ejemplo de este **(AgentState (temp 39) (battery 20)).**

En este sistema se tiene en cuenta la necesidad de crear un plan de juego que permita al equipo realizar una serie de estrategias, haciendo que los agentes se enfrenten a su oponente en forma organizada y siguiendo un plan que les permita hacer goles en el arco contrario, evitando que la pelota entre al arco propio. Para este propósito, es utilizado un lenguaje de descripción de la escena que no necesariamente está enlazado a un entorno físico real, pero que sí permite considerar las jugadas y condiciones pre-establecidas, implementando estrategias específicas para un dominio del juego. También permite hacer un análisis y modelado del adversario, lo cual lleva a tener un pre-aprendizaje de sus posibles jugadas y tácticas, llevando al equipo a optimizar su juego.

4. Experimentación

En este capítulo se describen los resultados conforme a los objetivos planteados, generados a partir de la realización del simulador FuroSimSpark. Se mostrará las capacidades que tiene el simulador, el paso a paso de una estrategia y como son las formaciones de los equipos.

Figura 4-1: FuroSimSpark ubicando el balón

4.1 Acciones y comportamientos

El simulador FuroSimSpark está programado para que cada elemento reproduzca diferentes aspectos físicos propios de un partido de fútbol cancha, jugadores, pelota, la resistencia al movimiento, desplazamiento, jugadas, etc. FuroSimSpark permite tener

control sobre los robots y sus jugadas, así como de la posición de la pelota y preveer los posibles movimientos de los robots contrarios.

Una vez cargados los dos equipos, el simulador inicia el partido que posteriormente se desarrollará en forma autónoma, salvo en los momentos que se determine como una pelota detenida, como saque de arco, penal, saque del medio, etc.

Las capacidades que el simulador FuroSimSpark permite mejorar son:

- Interactuar entre sí como equipo para desarrollar una labor conjunta (jugar).
- Hallar rutas cortas para moverse de un lado a otro.
- Modelar eficientemente las circunstancias que surgen para los jugadores durante un partido.
- Eficiencia y rapidez en la toma de decisiones por parte de los robots.
- Diseñar tácticas que les permita a los agentes jugar contra un adversario.
- Diseñar una conducta entre pase, tiro, lanzamiento, retención del balón y protección del arco.

Para lograr programar en el equipo estas y otras capacidades se deben programar en el simulador las siguientes actividades:

- Ubicar y moverse en la cancha.
- Ubicar la posición del balón.
- Trasladarse hacia el balón.
- Ubicarse cerca del balón.
- Ubicar la portería.
- empujar el balón hacia la portería.

4.2 Estrategias

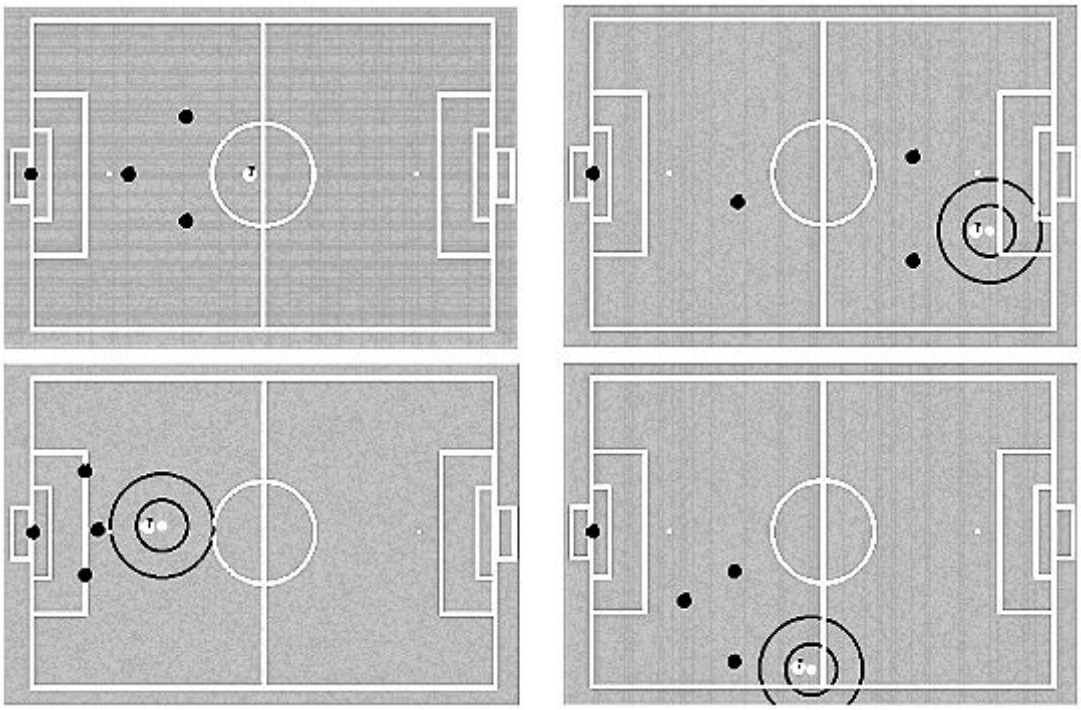
En el sistema multi-agente FuroSimSpark, cada robot es un agente independiente y coordina sus acciones con sus compañeros de equipo a través de la comunicación y el intercambio de información. El comportamiento individual debe integrarse en la estrategia global del equipo, dando como resultado acciones cooperativas de todos los agentes. Esto se hace mediante el uso de roles y comportamientos que se le define a cada agente, dependiendo de sus acciones individuales.

Como estrategia principal, los robots utilizan un modelo de coordinación implícito, basado en nociones del posicionamiento de los agentes, el rol que desempeña cada agente y su formación en el campo [22]. Todos estos elementos de información se dan en un archivo de configuración estratégica. En este archivo se define las diferentes posiciones y parámetros para el posicionamiento, la estructura de la defensiva, centrocampista y los movimientos estratégicos.

Durante el juego abierto, los agentes utilizan sólo tres roles así: un rol para el portero, uno para el centrocampista quien se mueve de acuerdo a su posición estratégica y uno para el delantero que podría ser el jugador activo y su posición debe ser la más cercana a la bola y al arco contrario listo para anotar.

Las formaciones son conjuntos de estrategias que definen un modelo de movimiento para los agentes jugadores, y a su vez cada posición es un modelo de movimiento para un jugador específico. La asignación de jugadores para una posición específica es dinámica, y se realiza de acuerdo a la posición de la bola, los compañeros y los jugadores adversarios [18], ver Figura 4-1.

Figura 4-1: Modelos de formación del equipo durante varias posiciones de la bola.



5. Conclusiones

Para este proyecto se modificó el simulador oficial de la Robocup 3D SimSpark siendo una buena opción para la generación de nuevos simuladores como el FuroSimSpark que fue adaptado a las necesidades del tipo de robots desarrollados por el grupo ALife, integrando inteligencia artificial y cierta libertad que permiten a los agentes jugar autónomamente.

El área de fútbol de robots involucra muchos temas como la robótica, la visión por computador, el reconocimiento de patrones en tiempo real, el planteamiento y coordinación de multi-agentes, entre otros. Son de interés y motivación en el campo académico, la investigación y el desarrollo, y permiten promover la integración y el desarrollo de nuevas tecnologías por parte de los interesados en este tipo de experiencias, dejando como en este caso material de consulta para quienes deseen dar sus primeros pasos dentro del ambiente de fútbol de robots.

El desarrollo de la mayoría de los simuladores está basado en ambientes ideales, haciendo que en un ambiente real las estrategias sean poco idóneas y funcionales.

Se hace necesario en el desarrollo de las estrategias hacer el respectivo análisis de los factores que intervienen durante el desarrollo del juego, de esta manera se pueden incorporar todas las variables para el desarrollo de las estrategias tanto en el ambiente simulado como en el real.

6. Trabajo Futuro

Se espera transferir fácilmente el conocimiento y resultados obtenidos de la investigación al comportamiento cooperativo y competitivo de los robots físicos durante el juego, añadir estrategias nuevas y que se puedan aplicar en proyectos relacionados con la robótica, la electrónica, la Ingeniería mecánica, la inteligencia artificial, etc.

A corto plazo se pretende mejorar el diseño del software, la arquitectura del hardware y los controladores, de tal manera que permitan la fusión de la información recogida por los sensores, optimizar el razonamiento y el control, desarrollando más estrategias sólidas e inteligentes de los agentes que les permitan jugar fútbol, resolviendo los problemas basados en la Inteligencia Artificial, la comunicación entre robots y el aprendizaje.

7. Bibliografía

[1] Baturone, A. O. (2005). Robótica: manipuladores y robots móviles. Marcombo.

[2]Kitano, H. (1998). RoboCup-97: robot soccer world cup I (Vol. 1). Springer Science & Business Media.

[3] Park, S. W., Kim, J. H., Kim, E. H., & Oh, J. H. (1997, April). Development of a multi-agent system for robot soccer game. In Robotics and Automation, 1997. Proceedings., 1997 IEEE International Conference on (Vol. 1, pp. 626-631). IEEE.

[4] Robotics and Automation Society, http://www.ieee-ras.org, página consultada Septiembre 2014

[5] International Federation of Robotics (IFR), http://www.ifr.org/, página consultada Septiembre 2015

[6] Bekey, George, Autonomous robots: from biological inspiration to implementation and control. MIT Press, 2005.

[7] Russell, Stuart y Norvig Peter. Artificial Intelligence: A Modern Approach. Prentice-Hall, Inc. 1995.

[8] Winston, Patrick. Artificial Intelligence. Third edition. Editorial Addison-Wesley Publishing Company. USA, May 1993.

[9] M. J. Wooldridge and N. R. Jennings, “Agent Theories, Architectures, and Languages: A Survey,” in Workshop on Agent Theories, Architectures & Languages (ECAI’94), ser. Lecture Notes in Artificial Intelligence, M. J. Wooldridge and N. R. Jennings, Eds., vol. 890. Amsterdam, The Netherlands: Springer-Verlag, Jan. 1995, pp. 1–22. [Online]. Available: citeseer.ist.psu.edu/article/wooldridge94agent.html

[10] H. Aberg. “Agent Roles in RoboCup Teams”. Tesis, Kungliga, Tekniska Högskolan School of Computer Science and Technology, 1998.

[11] Pedro U. Lima “Robotic Soccer” Institute for Systems and Robotics 2007

[12] Browning, B., &Tryzelaar, E. (2003). berSim : A Multi-Robot Simulator for Robot Soccer. Proceedings of the second international joint conference on Autonomous agents and multiagent systems, 948-949.ACM. Retrieved from http://portal.acm.org/citation.cfm?id=860739

[13] RoboCup, http://www.robocup.org/, página consultada 2014

[14] FIRA MiroSot Game Rules, http://fira.net/?mid=mirosot, página consultada 2014

[17] Rusell Smith, Open dynamics engine v0.8 user guide, ODE2007.

[18] Wang, Y. T., You, Z. J., & Chen, C. H. (2009). AIN-based action selection mechanism for soccer robot systems.Journal of Control Science and Engineering, 2009. doi:10.1155/2009/896310

[19] Chen, W. Y., &Payandeh, S. (2007). Micro Robot Hockey Simulator - Game Engine Design. 2007 IEEE Symposium on Computational Intelligence and Games (pp. 9-16). IEEE. doi:10.1109/CIG.2007.368073

[20] Teixeira, C., Lau, N., & Reis, L. P. (2004). FC Portugal 2003 Shoot Evaluation Based on Goalie Movement Prediction. Proceedings of Scientific Meeting of the Portuguese Robotics Open (pp. 149–155). Retrieved from http://paginas.fe.up.pt/~lpreis/Papers/FCPortugalShootingApproach.PDF

[21] Valente, P., Hossain, S., Gronbak, B., Hallenborg, K., & Reis, L. P. (2010). A multi-agent framework for coordination of intelligent assistive technologies. Information Systems and Technologies CISTI 2010 5th Iberian Conference on (pp. 1-6). IEEE. Retrieved from ttp://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5556631

[22] Boedecker, J., & Asada, M. (2008). SimSpark – Concepts and Application in the RoboCup 3D Soccer Simulation League. Autonomous Robots, 174-181. Retrieved from http://www.er.ams.eng.osaka-u.ac.jp/Paper/2008/Joschka08a.pdf

[23] Awaad, I. S. (2008). XPERSim: A Simulator for Robot Learning by Experimentation. Applied Sciences (Vol. 5325, pp. 5-16).Springer Berlin Heidelberg.doi:10.1007/978-3-540-89076-8

[25] RoboCup (2011). Robot Soccer, World Cup XV. (Vol. 7416, 2012, pp. 450-460)

[26] Guofeng, T., Hongsheng, H., &Zhenzhou, S. (2008). A HiFi simulator for Robot Soccer. 2008 7th World Congress on Intelligent Control and Automation (pp. 3045-3048). IEEE. doi:10.1109/WCICA.2008.4593407

[27]http://www.fing.edu.uy/inco/grupos/mina/pGrado/fibra/documents/Reglas%20para%20SimuroSot.pdf página consultada 2014

[28] Iocchi, L., Matsubara, H., Weitzenfeld, A., Zhou, C., Birk, A., Poppinga, J., Stoyanov, T., et al. (2009). RoboCup 2008: Robot Soccer World Cup XII. (L. Iocchi, H. Matsubara, A. Weitzenfeld, & C. Zhou, Eds.)Science And Technology, 5949, 463-472-472. SpringerBerlin Heidelberg. doi:10.1007/978-3-642-02921-9

[29] https://simtk.org/home/opensim página consultada 2014

[30] RoboCup 2004: Robot Soccer World Cup VIII UCHILSIM: A Dynamically and Visually Realistic Simulator for the RoboCup Four Legged League, (Volume 3276, 2005, pp 34-45)

[31] Larsen, E. (2001). A Robot Soccer Simulator: A Case Study for Rigid Body Contact. Methods, (March), 1-24. Retrieved from http://www.research.scea.com/research/pdfs/GDC2001robotsoccerERIC.pdf

[32] Michel, O. (2004). WebotsTM: Professional Mobile Robot Simulation. International Journal of Advanced Robotic Systems, 1(1), 39-42. Citeseer. Retrieved from http://arxiv.org/abs/cs/0412052

[33] Keene, (1988). A programmer's guide to object-oriented programming in common lisp, Addison-Wesley Longman Publishing Co., Inc. Boston, Ma, USA 1988.

[34] Peter Escamilla, Jonatan Gómez, (2011). FURO: Fútbol de robots, acercamiento a un ambiente cooperativo, software, hardware y agentes, 6th Colombian Computing Congress, Poster Session (2011)

[35] W. Sotomonte, “Estrategias de sistemas de agentes (simple y multiple): Caso de estudio futbol de robots,” Master’s thesis, Universidad Nacional.

[36]John McCarthy (1956), Conferencia de Darthmounth, Instituto de Tecnología de Massachusetts (MIT)

[37] Liu Leandro, Gomez Jonatan(2011), Reporte técnico fútbol de robots, Grupo Alife Universidad Nacional de Colombia.

[38] Siegwart, R (2011) et al. Introduction to autonomous mobile robots..

[39] Gopalakrishnana, B., S. Tirunellayi, and R. Todkar(2004), Design and development of an aoutonomous mobile smart vehicle: a mechatronics application.Mechatronics

[40] Kim, J.-H., & Vadakkepat, P. (2000). Multi-agent systems: a survey from the robot-soccer perspective. Intelligent Automation & Soft Computing, 6(1), 3–17.

8. Anexo

8.1 Modelo Robot Furo

```
; -*- mode: lisp; -*-
; this file constructs a newfuro, the connected agent controlls
; two joint motor
(RSG 0 1)
(
  (def $lenXY       0.75)
  (def $lenZ       0.7)
  (def $totalMass      1000)
  (nd Space
    (setName Furo)
    (disableInnerCollision true)
    (nd AgentAspect
      (setName body)
      (setLocalPos 0 0 (eval 0.15 * $lenZ  ))
      (nd StaticMeshInitEffector)
      (nd TimePerceptor)
      (nd RigidBody
         (setName boxBody)
         (setBox (eval 0.96 * $totalMass ) $lenXY $lenXY $lenZ)
         (nd DragController
           (setAngularDrag 0.01)
           (setLinearDrag 0.01)
        )
      )
      (nd StaticMesh
         (load 'models/Furobox.obj')
         (setScale (eval 0.5 * $lenXY  ) (eval 0.5 * $lenXY) (eval 0.5 * $lenZ  ))
```

```
  )
    (nd BoxCollider
       (setBoxLengths $lenXY $lenXY $lenZ)
       (nd ContactJointHandler
         (setContactBounceMode false)
         (setContactSlipMode true)
         (setContactSlip 0.1 0.1)
         (setContactSoftERPMode true)
         (setContactSoftERP 0.2)
         (setContactSoftCFM true)
         (setContactSoftCFM 0.01)
       )
     )
  (nd AgentState
    (setName AgentState)
     ; add a gamestateperceptor
    (nd GameStatePerceptor)
    (setRobotType 0)
    (nd Transform
      (nd Cylinder
        (setName SelectionMarker)
        (setParams 1.0 1.0)
        (setScale 0.2 0.2 0.02)
        (setMaterial matSelect)
        (setTransparent)
      )
    )
  )
  (nd GyroRatePerceptor (setName torso))
  (nd Accelerometer (setName torso))
  (nd BeamEffector)
  (nd SayEffector)
  (nd AgentSyncEffector)
  (nd VisionPerceptor
    (setSenseMyPos true)
    (setStaticSenseAxis false)
    (addNoise false)
  )
);end of AgentAspect
```

```
    (nd Transform
      (setLocalPos (eval 0.5 * $lenXY  ) 0 0)
      (setName r1)
      (setLocalRotation 0 90 0)
        (importScene rsg/boxspheres/wheel.rsg (eval 0.35 * $lenZ  ) (eval 0.2 * $lenXY  )
(eval 0.02 * $totalMass ) matDarkGrey)
      (nd HingeJoint
        (attach ../sphereBody ../../body/boxBody)
        (setAnchor 0.0 0.0 0)
        (setAxis 2) ; move around z-axis
        (setMaxMotorForce 0 400)
        (setJointMaxSpeed1  200)
        (nd HingePerceptor   (setName sr1) )
        ; install an effector to control the joint motors
        (nd HingeEffector (setName er1) )
      )
    )
    (nd Transform
      (setLocalPos (eval -0.5 * $lenXY  ) 0 0)
      (setName r3)
      (setLocalRotation 0 90 0)
         (importScene rsg/boxspheres/wheel.rsg (eval 0.35 * $lenZ  ) (eval 0.2 * $lenZ  )
(eval 0.02 * $totalMass ) matDarkGrey)
      (nd HingeJoint
        (attach ../sphereBody ../../body/boxBody)
        (setAnchor 0.0 0.0 0)
        (setAxis 2) ; move around z-axis
        (setMaxMotorForce 0 400)
        (setJointMaxSpeed1  200)
        (nd HingePerceptor   (setName sr3) )
        ; install an effector to control the joint motors
        (nd HingeEffector (setName er3) )
      )
    )
  );end of nd Space
)
```

8.2 Control Manual.java (metodos y elementos principales)

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */

/*
 * ControlManual.java
 *
 * Created on 27/09/2011, 12:56:02 AM
 */

package Control;

import Serial.rxtx;
import Serial.rxtx.SerialWriter;
import com.centralnexus.input.Joystick;
import com.centralnexus.input.JoystickListener;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.IOException;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.swing.text.DefaultCaret;

/**
 *
 * @author Leandro Liu
 */
public class ControlManual extends javax.swing.JFrame
 implements Runnable, JoystickListener, ActionListener
{

        private Joystick joy;
```

```
private Joystick joy2;


private rxtx serial;
private SerialWriter sw;
private int mode=0;
private inercialControl iC;
private boolean keyed=false;
private char lastKey;
private char topSpeed;
private char minSpeed;
private char d;

private Thread thread;
private int joyButtonsold=0;
private int joy2Buttonsold=0;

/** Creates new form ControlManual */
public ControlManual() throws InterruptedException, IOException {
initComponents();
thread = new Thread(this);

DefaultCaret caret = (DefaultCaret)ConsolaTxt.getCaret();
caret.setUpdatePolicy(DefaultCaret.ALWAYS_UPDATE);
iC=new inercialControl();

joy = Joystick.createInstance();
for (int idx = joy.getID() + 1; idx < Joystick.getNumDevices(); idx++) {
if (Joystick.isPluggedIn(idx)) {
        joy2 = Joystick.createInstance(idx);
}
}
if (joy2 == null) {
joy2 = joy;
}



}
```

```
        private void sendBtnMouseClicked(java.awt.event.MouseEvent evt)
{//GEN-FIRST:event_sendBtnMouseClicked
        if(sw!=null){
        sw.send(manualTxt.getText());
        manualTxt.setText("");
        }else{
        System.out.println("Err: no hay serial Writer");
        }
        }//GEN-LAST:event_sendBtnMouseClicked

        private void BtnDirectoMouseClicked(java.awt.event.MouseEvent evt)
{//GEN-FIRST:event_BtnDirectoMouseClicked
        if(this.mode==1){
        this.BtnDirecto.setText("Activar");
        this.mode=0;
        }else{
        this.BtnDirecto.setText("Desactivar");
        this.mode=1;
        }
        }//GEN-LAST:event_BtnDirectoMouseClicked


        private void sendBtnActionPerformed(java.awt.event.ActionEvent evt)
{//GEN-FIRST:event_sendBtnActionPerformed
        // TODO add your handling code here:
        }//GEN-LAST:event_sendBtnActionPerformed

        private void sValBtnMouseClicked(java.awt.event.MouseEvent evt)
{//GEN-FIRST:event_sValBtnMouseClicked
        if(sw!=null){
        int a=Integer.parseInt(manualTxt.getText());
        String s;
        if(a>127){
                //System.out.println(a);
                char b=(char)(a-128);
                b+=128;
                s=Character.toString(b);
```

```
                System.out.println(s.length());

        }else{
                s=Character.toString((char) a);
        }
        sw.send(s);
        System.out.println(s.getBytes());
        manualTxt.setText("");
        }else{
        System.out.println("Err: no hay serial Writer");
        }
        }//GEN-LAST:event_sValBtnMouseClicked

        private void portTxtActionPerformed(java.awt.event.ActionEvent evt)
{//GEN-FIRST:event_portTxtActionPerformed
        // TODO add your handling code here:
        }//GEN-LAST:event_portTxtActionPerformed

        private void formKeyReleased(java.awt.event.KeyEvent evt)
{//GEN-FIRST:event_formKeyReleased
        this.keyed=false;
        System.out.println("formKeyReleased");

        }//GEN-LAST:event_formKeyReleased

        private void TxtVelocidadKeyPressed(java.awt.event.KeyEvent evt)
{//GEN-FIRST:event_TxtVelocidadKeyPressed
        if((this.TxtVelocidad.getText()).length()>0)
        this.topSpeed= (char)Integer.parseInt(TxtVelocidad.getText());
        }//GEN-LAST:event_TxtVelocidadKeyPressed

        private void TxtAceleracionKeyPressed(java.awt.event.KeyEvent evt)
{//GEN-FIRST:event_TxtAceleracionKeyPressed
        if((this.TxtAceleracion.getText()).length()>0)
        this.minSpeed=(char)Integer.parseInt(TxtAceleracion.getText());
        }//GEN-LAST:event_TxtAceleracionKeyPressed

        private void jPanel1KeyPressed(java.awt.event.KeyEvent evt)
{//GEN-FIRST:event_jPanel1KeyPressed
```

```
        // TODO add your handling code here:
        }//GEN-LAST:event_jPanel1KeyPressed

        private void sValBtnActionPerformed(java.awt.event.ActionEvent evt)
{//GEN-FIRST:event_sValBtnActionPerformed
        // TODO add your handling code here:
        }//GEN-LAST:event_sValBtnActionPerformed

        public void dataReceived(String msg){
        ConsolaTxt.append(msg);

        }
        public void setOutput(SerialWriter sw){
        this.sw=sw;
        }
        /**
        * @param args the command line arguments
        */
        public static void main(String args[]) {
        java.awt.EventQueue.invokeLater(new Runnable() {
        public void run() {
                try {
                ControlManual mainFrame=new ControlManual();
                mainFrame.setVisible(true);
                mainFrame.startPolling();
                } catch (InterruptedException ex) {
                Logger.getLogger(ControlManual.class.getName()).log(Level.SEVERE,
null, ex);
                } catch (IOException ex) {
                Logger.getLogger(ControlManual.class.getName()).log(Level.SEVERE,
null, ex);
                }
        }
        });
        }

        private javax.swing.JButton BtnDirecto;
        private javax.swing.JButton BtnFija;
        private javax.swing.JButton BtnInercial;
```

```
        private javax.swing.JTextArea ConsolaTxt;
        private javax.swing.JTextField TxtAceleracion;
        private javax.swing.JTextField TxtObjetivo;
        private javax.swing.JTextField TxtVelocidad;
        private javax.swing.JButton connectBtn;
        private javax.swing.JLabel jLabel1;
        private javax.swing.JLabel jLabel2;
        private javax.swing.JLabel jLabel3;
        private javax.swing.JLabel jLabel4;
        private javax.swing.JLabel jLabel5;
        private javax.swing.JLabel jLabel6;
        private javax.swing.JLabel jLabel7;
        private javax.swing.JPanel jPanel1;
        private javax.swing.JPanel jPanel2;
        private javax.swing.JPanel jPanel3;
        private javax.swing.JPanel jPanel4;
        private javax.swing.JPanel jPanel5;
        private javax.swing.JPanel jPanel6;
        private javax.swing.JPanel jPanel7;
        private javax.swing.JScrollPane jScrollPane1;
        private javax.swing.JTextField manualTxt;
        private javax.swing.JTextField portTxt;
        private javax.swing.JButton sValBtn;
        private javax.swing.JButton sendBtn;


        public void joystickAxisChanged(Joystick j) {

        if (j == joy) {
        updateFieldsEx(j);
        }
        else {
        updateFields(j);
        }
        }

        public void joystickButtonChanged(Joystick j) {
System.out.println("YY"+j.getButtons());
```

```
        if (j == joy) {
        updateFieldsEx(j);
        }
        else {
        updateFields(j);
        }
        }

        public void actionPerformed(ActionEvent e) {
        joy.addJoystickListener(this);
        }
public void updateFields(Joystick joystick) {


        int joy2Buttons=joystick.getButtons();


        if(joy2Buttonsold!=joy2Buttons){
        joy2Buttonsold=joy2Buttons;
        System.out.println("Fields_ "+Integer.toHexString(joystick.getButtons()));

        switch(joy2Buttonsold){
        case 1:sw.send("q");break;
        case 2:sw.send("n");break;
        case 4:sw.send("e");break;
        case 8:sw.send("b");break;
        case 16:sw.send("s");break;
        }

        }


        }

        public void updateFieldsEx(Joystick joystick) {

        int joyButtons=joystick.getButtons();
```

```
        if(joyButtonsold!=joyButtons){
        joyButtonsold=joyButtons;
        System.out.println("FieldsEx_ "+Integer.toHexString(joystick.getButtons()));
        byte a=(byte)joy2Buttonsold;

        sw.send((byte)0x0 );
        sw.send(a );


        }
        }
        public void startPolling() {
        thread.start();
        // this.run();

        }

        public void run() {
        for (;;) {
        joy.poll();
        joy2.poll();

        updateFieldsEx(joy);
        updateFields(joy2);
        try {
                Thread.sleep(50);
        } catch(InterruptedException e) {
                break;
        }
        }
        }
}
```