CENTRO DE INVESTIGACIÓN Y DE ESTUDIOS AVANZADOS
DEL INSTITUTO POLITÉCNICO NACIONAL

UNIDAD ZACATENCO
DEPARTAMENTO DE COMPUTACIÓN

Simulación de comportamiento para robots humanoides en un juego de fútbol

Tesis que presenta

Luis Enrique Figueroa Medina

Para obtener el grado de

Maestro en Ciencias
en Computación

Director de la Tesis: Dr. Pedro Mejía Álvarez

Codirector de la Tesis: Dr. Juan Manuel Ibarra Zannatha

México, D.F. Enero 2011

Los abajo firmantes , integrantes del jurado de examen de grado que sustentara el Sr. Luis Enrique Figueroa Medina, declaramos que hemos revisado la tesis titulada:

Simulación de comportamiento para robots humanoides en un juego de fútbol

Y consideramos que cumple los requisitos para obtener el Grado de Maestría en Ciencias en Computación, firmamos la presente en la Ciudad de México, D.F., el mes de Enero de 2011.

Atentamente

Dr. Pedro Mejía Álvarez ----------

Dr. Juan Manuel Ibarra Zannatha ----------

Dra. Sonia G. Mendoza Chapa ----------

Dr. Adriano de Luca Pennacchia ----------

Primeramente le agradezco a Dios por permitirme estar con vida

y enfrentar cada reto con pasión y devoción.

Agradezco a mis padres Facundo Enrique Figueroa Bautista

y María Medina Guerrero por mostrarme el camino a

seguir toda la vida y estar apoyándome siempre

en los momentos más difíciles de mi vida

Agradecimientos

A Dios por mostrarme el camino a seguir durante mi vida.

A mis padres por su amor y apoyo incondicional, por ser las fuerzas que me ayudan a seguir.

A mis hermanas, Marisol y Carol por sus palabras de aliento en los momentos más difíciles.

A mis sobrinos Héctor y Lesslie, que me verán de ejemplo y por ayudarme a ser mejor persona.

Al Dr. Pedro Mejía Álvarez por su paciencia y motivación en le desarrollo de esta tesis. Por ser una persona admirable y sabia, y que siempre me regalo el tiempo necesario.

Al Dr. Juan Manuel Ibarra Zannatha por enseñarme todo un mundo nuevo y desconocido… la robótica.

A mis revisores Dra. Sonia Mendoza Chapa y Dr. Adriano de Luca por brindarme su tiempo en la revisión de mi tesis y tener el conocimiento para ayudarme a finalizarla.

Al Dr. Dominique Decouchant por ser mi tutor en mi primer año de maestría y por alentarme a salir adelante.

Al Mtro. Javier Valle Mora por motivarme, ayudarme con su enseñanza cuando mas lo necesitaba y regalarme parte de su tiempo… muchas gracias.

Al CINVESTAV por permitirme formar parte de una institución reconocida mundialmente.

Al CONACyT por el apoyo económico brindado a lo largo de mi estancia en el CINVESTAV

A mis amigos del Departamento de Computación, y con afecto a Juan, JuanK, Tanibet, Iván, Edgardo y Carlos.

A mis amigos de Control Automático y La-Salle, y con afecto a Rafael, Héctor, David, Jorge, Erick, Rubén y los Padawan.

Índice general

Índice Figuras

Índice Tablas

Resumen

La robótica se encuentra actualmente como un área de la tecnología que permite automatizar muchos procesos del ámbito industrial. Así mismo, los robots actualmente son utilizados para emular actividades de la vida humana tales como la limpieza de plantas industriales y edificios, la vigilancia de aeropuertos y edificios o el cuidado de enfermos terminales, por solo mencionar algunas. En este proyecto de tesis, pretendemos investigar sobre esta capacidad de emulación humana de los robots y aplicarla para enseñarlos a jugar fútbol de forma sincronizada, coordinada y en equipo con otros robots.

Específicamente, en este trabajo se desarrollará un simulador del juego de fútbol para robots humanoides. Se pretende integrar a este simulador técnicas de Inteligencia Artificial a fin de que los robots integrantes del equipo jueguen de forma inteligente e intenten ganar su juego. De esta forma, y a partir de la inteligencia producida en el simulador sobre las jugadas, se pretende transmitir estas capacidades a los robots humanoides físicos a fin de que mejoren su juego.

El uso del simulador pretende también ser una herramienta para enseñar el juego a los robots, construir jugadas, mejorar la planeación de los movimientos, el sensado del ambiente, y la dinámica de los movimientos de las distintas articulaciones del robot.

Abstract

Robotics is currently an area of technology to automate many processes in the industrial field. Also, robots are currently used to emulate human life activities such as cleaning of industrial plants and buildings, airports and buildings monitoring or hospice care, to name a few. In this thesis, we intend to investigate this human emulation capability of robots and apply it to teach them to play soccer in synchronization and coordination with other robots.

Specifically, this work focuses on developing a football soccer game simulator for humanoid robots. It intends to integrate this simulator Artificial Intelligence techniques to the robot team members, to play intelligently and try to win their game. In this way, and from the intelligence produced in the simulator on the plays, it is intended to convey these skills to physical humanoid robots in order to improve their game.

The use of the simulator is also intended as a tool to teach the game to the robots, to build a play, to improve the planning of movement, sensing the environment, and the dynamics of the movements of the joints of the robot.

Capítulo 1

Introducción

1.1. Antecedentes

En el Laboratorio de Robótica y Visión Artificial del Depto. de Control Automático del Cinvestav-México se ha estado trabajando recientemente en el desarrollo de modelos geométricos, cinemáticos y dinámicos de robots humanoides, así como de sistemas de percepción, programación, control de Robots Humanoides a fin de que estos puedan desempeñar tareas tales como jugar fútbol dentro de un equipo.

El objetivo de este proyecto es que estos robots sean capaces de reconocer su medio-ambiente y que jueguen en coordinación con otros robots como equipo en contra de un adversario. En el último año, el laboratorio ha sido capaz de programar un equipo de robots para integrarlo al torneo mundial de Fútbol para robots conocido como RoboCup, el cual se llevó a cabo en Graz, Austria en el verano de 2009.

Hasta ahora, se han utilizado robots comerciales de pequeño formato y bajo costo que, si bien permiten adentrar en esta temática con cierto nivel, para los actuales objetivos y reglas que exige este torneo y considerando las experiencias que se han acumulado, se hace necesario contar con una infraestructura de hardware y software de mayor nivel competitivo.

Actualmente los robots utilizados tienen poca capacidad de cómputo, baja memoria y muy poca inteligencia.

Por estas razones, se ha iniciado un proyecto para desarrollar una nueva generación de RHuA (Robots Humanoides Autónomos) que permitan mejorar el nivel de juego de los robots y sus capacidades de juego inteligente.

Dicho proyecto involucra el desarrollo de una nueva estructura de hardware y mecánica, de nuevos actuadores, de un sistema de cómputo mucho más potente, de un sistema de percepción más completo, y de un nuevo enfoque de programación inteligente de los robots basada en agentes que incluya interfaces amigables con el usuario y sistemas de simulación más potentes.

1.2 Motivación

Uno de los temas de investigación con un mayor interés tanto científico como tecnológico dentro de la Robótica es el de los Robots Humanoides Autónomos. No sólo se trata del desarrollo de una mecánica que permita al robot caminar en postura erguida y realizar movimientos similares a los de un ser humano, sino también todos los problemas electromecánicos ligados a la actuación de los mecanismos que permiten el movimiento y su autonomía. De la misma forma, este problema involucra a actividades tales el modelado, la simulación y el control inteligente de un complejo mecanismo robótico, el cual cuenta con más de 20 grados de libertad.

> La organización RoboCup viene realizando competencias de fútbol entre equipos de robots humanoides autónomos a nivel mundial desde hace 10 años. Cada año esta competencia Robótica ha incrementado significativamente las restricciones a que se ven sometidos los robots participantes para tratar de que se parezcan cada vez más a los fútbolistas humanos y que jueguen con reglas cada vez más parecidas a las de FIFA.

1.3 Planteamiento del problema

El objetivo general de esta tesis consiste en programar un Simulador de Juego Robótico a fin de que realice las siguientes actividades:

1. Mejorar la capacidad de movimiento, ir de una lado a otro más rápido y siguiendo el camino más corto.
2. Mejorar la capacidad de la percepción y sensado del medio ambiente: poder reconocer de forma rápidamente el entorno a fin de tomar mejores decisiones en tiempo real.
3. Poder modelar de forma más eficiente los distintos estados por los que pasa el jugador durante el juego.
4. Mejorar la capacidad de cada robot para pasar de un estado a otro de forma que se logren los objetivos de eficiencia y rapidez.
5. Diseñar mecanismos inteligentes que permitan a los robots diseñar estrategias para jugar al fútbol contra un adversario.

En esta tesis se pretende programar a un solo robot para que lleve a cabo una secuencia de actividades de Juego de Fútbol Robótico. Se pretende desarrollar modelos orientados a objetos para eficientar los movimientos y las actividades del robot. Las actividades que se pretende programar en simulación en esta tesis son las siguientes:

1. Levantarse del suelo.
2. Ubicar su posición en la cancha.
3. Ubicar la posición del balón.
4. Caminar hacia el balón.
5. Ubicarse cerca del balón.
6. Ubicar la portería.
7. Patear el balón hacia la portería.

1.4 Objetivos generales y específicos del proyecto

El objetivo general de esta tesis consiste en programar la simulación mediante Webots de un Juego Robótico de Fútbol bajo las reglas del torneo RoboCup.

Los objetivos particulares son los siguientes:

1. Percepción (obtención de información visual)
    a. Detección de la portería.
    b. Detección de la pelota.
    c. Auto-localización.

2. Movimientos (envió de señales a los motores).
    a. Implementación de locomociones nuevas.
    b. Optimización de las locomociones por defecto.

3. Control de movimientos y comportamiento
    a. Estrategia de tiro.
    b. Estrategia de caminado.
    c. Estrategia de acomodamiento.

1.5 Infraestructura

En la infraestructura se requieren los siguientes materiales para desarrollar el proyecto de tesis:

1. Software comercial: Webots
2. Lenguaje de programación: C++
3. PC con requerimientos mínimos: 1.0 GHz, 128 tarjeta de video (NVIDIA), 256 Mb en RAM, Disco duro de 20 Gb.
4. Sistema Operativo: Windows, Mac o Linux.

1.6 Contribuciones o resultados esperados

Se espera como resultado un simulador que sea capaz de:

- Simular el comportamiento del robot.
- Añadir estrategias nuevas
- Ser compatible con el robot físico
- Participar en el torneo de la RoboCup en la categoría de simuladores.

1.7 Descripción de los capítulos

En el capitulo 2 se detalla los conceptos básicos de los robots, su clasificación, las características para identificar si un robot es humanoide o bípedo, la autonomía y la locomoción de un robot. También se presentan los conceptos sobre Inteligencia Artificial y Maquina de Estados Finitos. También en este capitulo se menciona la competencia RoboCup, las ligas en las que se divide, el robot Nao que se utiliza para la competencia RobotStadium y una introducción al simulador Webots.

En el capítulo 3 se detallan algunos de los distintos simuladores que existen en la competencia de RoboCup, así como las características principales de cada uno de ellos. Se menciona también los diferentes robots comerciales que se usan para otras categorías, así como los que se utilizaban para la categoría de plataforma estándar. Además, se menciona el trabajo del robot Nao en diferentes partes del mundo ya sea de forma virtual o de forma física.

En el capítulo 4 se introduce el tema de la simulación robótica, así como los detalles de la programación del simulador Webots y su arquitectura.

En el capitulo 5 se discute la simulación del juego Robótico de Fútbol utilizando Webots. Este capitulo se describe la estructura de la programación de robot Nao sobre una plataforma de desarrollo. Se menciona también como manipular los sensores y actuadores del Nao en el simulador de Webots, la programación de las secuencias de juego en el simulador y por ultimo la inteligencia artificial que se usa para controlar al robot Nao en esta plataforma.

En el capítulo 6 se describen los logros y el trabajo de tesis.

Finalmente en el capítulo 7 se mencionan las conclusiones que se obtuvieron en el desarrollo de este trabajo de tesis.

Capitulo 2

Conceptos básicos de la Robótica

2.1. Robot

Según la *International Federation of Robotics (IFR)* un robot, también llamado sistema robótico o manipulador, es una máquina de manipulación automática reprogramable capaz de posicionar y orientar materiales, piezas, herramientas o dispositivos especiales para la ejecución de trabajos diversos en las diferentes etapas de la producción industrial [1] (ya sea en una posición fija o en movimiento). En esta definición, la idea clave es la reprogramabilidad. Usualmente una máquina ordinaria se diseña con un grado de libertad que permita llevar a cabo todos los movimientos pre-programados mediante un sólo sistema actuador (sea éste un motor). Sin embargo, un robot se diseña con más de un grado de libertad, con el fin de lograr con ello una automatización flexible y así facilitar la programación del movimiento deseado según las necesidades cambiantes de la industria (Figura 1). Un grado de libertad se refiere a la capacidad que tiene un manipulador de realizar movimientos, ya sea horizontales, verticales o angulares.

Figura 1. Robot con tres grados de libertad.

2.1.1. Robots móviles

Básicamente existen dos grandes áreas de investigación en Robótica: la manipulación y la movilidad [2]. Sin embargo, estas dos áreas no son excluyentes entre sí, sino que se encargan de distintos aspectos de un sistema robótico. Esto es, un robot manipulador puede estar anclado a un punto específico de la línea de producción o bien, puede estar montado sobre un robot móvil. No obstante, el problema de manipulación y todos los aspectos que este conlleva

son los mismos en ambos casos. Esto implica posicionar, orientar y mover el efector final con un determinado patrón de velocidad sobre su espacio de trabajo, así como ejercer un control de fuerza cuando interacciona con su ambiente. Así mismo, el problema de movilidad es el mismo si un robot manipulador se encuentra montado o no sobre un robot móvil. En este caso el problema es el de la percepción del entorno, la localización, la navegación y el control de movimiento del robot. De esta manera es posible abordar cada área de investigación de manera independiente.

Un robot móvil [3] es aquel robot capaz de moverse en un cierto entorno mediante algún medio de locomoción, con el objetivo de realizar alguna tarea que requiera de dicha movilidad, presentando para ello un cierto grado de autonomía en su comportamiento (Figura 2). En la anterior definición se observan dos características importantes de un sistema robótico móvil: autonomía y locomoción. Cada una de ellas da origen a diferentes taxonomías de robots móviles, la cuales se describen a continuación.

Figura 2. Robot movil de seis ruedas con diferentes grados de libertad.

2.1.2. Autonomía

La autonomía en un sentido amplio se refiere a la capacidad que poseen ciertos sistemas de operar en ambientes del “mundo real”. Esta capacidad permite llevar a cabo tareas de manera deliberada y exhibir comportamientos inesperados propiciados por estímulos del ambiente, denominados comportamientos reactivos. Todo esto sin requerir alguna forma de control externo por parte de un operador por períodos extendidos de tiempo [4]**.**

Hoy en día, la mayoría de los robots móviles no son completamente autónomos, a menos que el entorno se encuentre bajo control. Tal es el caso de

los robots móviles de servicio que operan en estructuras creadas por el hombre (como edificios, calles, etc.). Otros robots móviles destinados a trabajar en entornos no controlados, generalmente no creados por el hombre (bosques, terrenos rocosos como Marte, ambientes acuáticos, etc.) son aún incapaces de mostrar autonomía en cuanto a cognición se refiere. Estos requieren de la intervención del ser humano en actividades como localización y navegación, y suelen ser tele-operados. Sin embargo estos robots presentan un control autónomo en sus movimientos de locomoción para efectuar la tarea de navegación requerida, por lo que aún en estos casos presentan un cierto grado de autonomía.

Existen distintos grados de autonomía de un robot [4]:

- **Nivel de control bajo**. Se refiere al control a nivel de articulación, el cual se logra con las especificaciones de diseño requeridas.

- **Nivel de control intermedio.** Pertenece al control de navegación, el cual trata de evitar colisiones entre los robots, o bien con cualquier obstáculo. Se espera que los robots puedan ejecutar las tareas específicas para las cuales fueron diseñados, al mismo tiempo que deben navegar en el entorno recibiendo información de sensores. Los robots móviles actuales se pueden considerar autónomos en este nivel de control, aunque todavía requieren de la intervención de un operador humano.

- **Nivel de control alto**. Representa un control a nivel tarea del robot. Estos son especificados en un lenguaje natural, a fin de llevar a cabo la planeación de la navegación. Actualmente ya existen robots autónomos a este nivel de control, como los que juegan algún deporte como el fútbol soccer.

2.1.3. Locomoción

La locomoción en biomecánica, se refiere a la manera en que los sistemas biológicos se mueven. Es decir, se refiere a los mecanismos que utilizan para realizar un movimiento no acotado que depende del ambiente para realizar su trayectoria [3].

Un mecanismo de locomoción consta de tres elementos principales: la fuente de potencia, el sistema de transmisión y las salidas de potencia. La fuente de potencia en un sistema biológico se localiza en los músculos del organismo, donde la energía química se convierte en energía mecánica. El sistema de transmisión transfiere la energía mecánica producida en la fuente de potencia a las salidas de potencia. Estas últimas corresponden a las partes del organismo

que se encuentran en contacto directo con el ambiente y que producen su movimiento [5].

Los sistemas biológicos de locomoción, que han evolucionado en la naturaleza, muestran un buen desempeño en términos de estabilidad, capacidad de carga y comportamiento dinámico [5]. Es por ello que estos sistemas representan la principal fuente de inspiración en el diseño de mecanismos de locomoción para robots móviles.

No obstante, la replicación fiel de los mecanismos de los sistemas biológicos no es nada sencillo; al contrario, es demasiado complejo. Las células representan bloques estructurales microscópicos especializados que, en conjunto, forman mecanismos de locomoción complejos, que solo se han podido aproximar hasta ahora mediante elementos estructurales macroscópicos simples [3]. Estos elementos forman una estructura cinemática simplificada basada en eslabones y articulaciones de un grado de libertad controlada por actuadores, que solo asemejan someramente la funcionalidad de dichos sistemas biológicos. Por ejemplo, huesos, articulaciones óseas y músculos generalmente se modelan como eslabones, pares cinemáticas y actuadores en una estructura cinemática.

Una clasificación natural de los robots móviles, se basa en el mecanismo de locomoción empleado de acuerdo a un entorno o ambiente en el que se realiza el movimiento. En la literatura [4] se proponen cuatro principales medios en los que se puede efectuar el movimiento de un robot móvil:

1. **Terrestre**. Los robots desarrollados para este entorno se fundamentan en conceptos de mecánica clásica de sólidos. Esto ha dado origen a muchas alternativas de desplazamiento que van desde las convencionales basadas en ruedas, hasta modelos de imitación del mundo animal basados en patas o piernas, como es el caso de los seres humanos (bípedos) o el caso de los demás mamíferos (cuadrúpedos), algunos insectos (hexápodos) y arácnidos (octópodos), o bien basados en movimientos reptantes (serpientes). Todos estos mecanismos de locomoción permiten el movimiento del robot sobre la superficie terrestre. Sin embargo, existen otros mecanismos de locomoción menos usuales inspirados en la naturaleza que permiten el desplazamiento mediante saltos utilizando algún mecanismo de propulsión, o el desplazamiento sobre paredes o superficies inclinadas. Incluso hay mecanismos inspirados en entornos arbóreos como los primates cuando se desplazan de rama en rama (Figura 3).

2. **Acuático**. Los robos móviles acuáticos se subdividen en aquellos que se mueven sobre la superficie del agua y los subacuáticos, cuyo movimiento es por debajo. Sobre la superficie del agua se encuentran los robots tipo

lancha que se impulsan generalmente mediante hélices o turbinas. Los robots subacuáticos también suelen utilizar hélices, sin embargo también hay otros que buscan imitar el comportamiento de animales acuáticos mediante el empleo de aletas o movimientos ondulatorios. En todos estos casos, el diseño se concentra en conceptos de hidrodinámica (Figura 4).

3. **Aéreo**. Ambiente explotado en investigaciones para aplicaciones militares y climatológicas. Entre los robots móviles aéreos se pueden encontrar los vehículos aéreos autónomos como aviones, helicópteros, globos aerostáticos y dirigibles. Sin embargo, en este caso se ha buscado reproducir el comportamiento de los insectos mediante la implementación del aleteo como medio de locomoción aéreo (ornicópteros). Pero sin importar cuál sea el mecanismo de locomoción empleado en este entorno, el correspondiente diseño se debe fundamentar en conceptos de aerodinámica, por lo que su implementación es difícil (Figura 5).

4. **Espacial**. En este medio se agrupan los robots que navegan en el espacio exterior. Aquí los robots utilizan medios de propulsión especiales como la propulsión nuclear, iónica, de combustible sólido. Navegan a partir de cálculos gravitacionales; esta clase de robots cuenta con un grado mucho mayor de autonomía debido a los retrasos de transmisión y recepción de datos, que se incrementan a medida que el robot se aleja de la tierra (Figura 6).

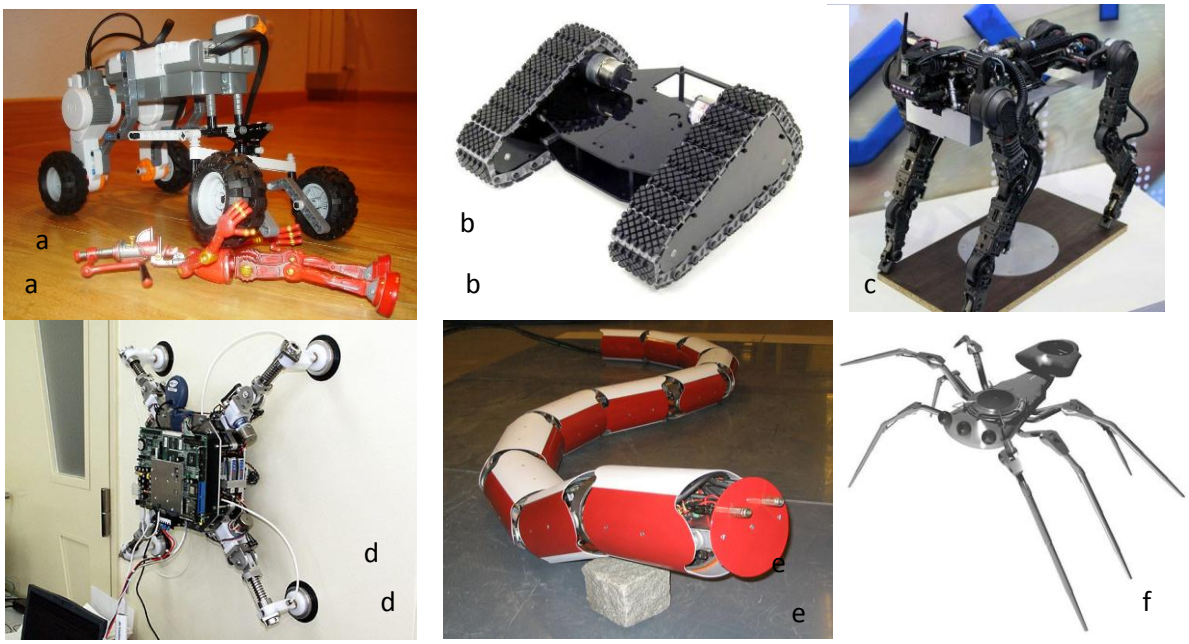


Figura 3. Tipos de robots terrestres: a) de rueda, b) de orugas, c) cuadrúpedo, d) escalador, e) movimientos ondulantes y f) octópodo

Figura 4. Tipos de robots móviles acuáticos: a) que se mueve sobre la superficie. b) que se mueve debajo de la superficie

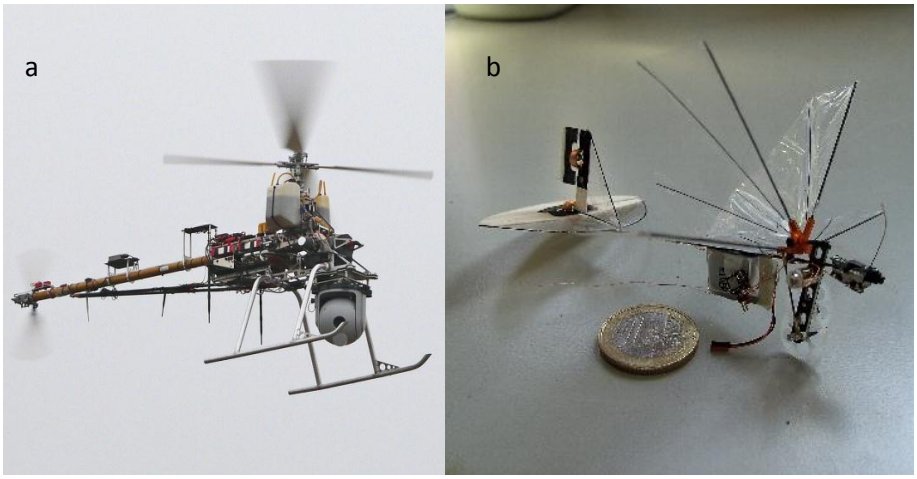


Figura 5. Tipos de robots móviles aéreos: a) helicóptero y b) ornicóptero

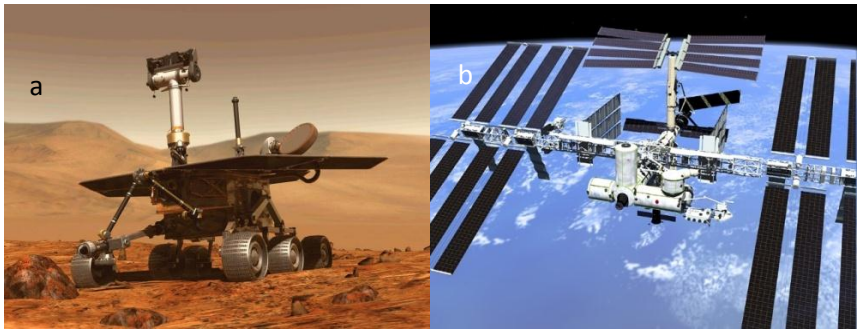


Figura 6. Tipos de robos espaciales.
a) robot en Marte y b) robot en la atmosfera.

De todos estos robots, los más comunes son los terrestres que utilizan ruedas como mecanismos de locomoción, aunque cada vez son más populares los robots caminantes que utilizan extremidades para tal efecto. Como ya se mencionó previamente, las ruedas además de ser extremadamente eficientes sobre suelo plano son mecánicamente sencillas. Por otro lado el mecanismo de locomoción utilizado por los robots caminantes es más complejo. En este caso el control de sus movimientos tiene un grado de dificultad que crece con el número de extremidades o patas, pues aumenta el número de combinaciones entre los posibles movimientos del robot [6].

La eficiencia de los robots móviles terrestres con ruedas depende en gran medida del entorno de operación del robot. Los tipos de entorno pueden ser, planos y estructurados o irregulares y no estructurados. Y los entornos pueden tener distinta dureza. Por otro lado, la eficiencia de los robots caminantes no depende del entorno, sino de su estructura mecánica. Es por ello que los robots con ruedas son más eficientes que los robots caminantes en entornos planos, pero no así en entornos irregulares [3]. Por esto, un robot caminante puede moverse de manera estable sobre superficies muy irregulares en donde existan obstáculos de todos los tamaños. Este tipo de robot es capaz de realizar movimientos omnidireccionales holonómicos sin deslizamiento y sin dañar la superficie del suelo [6].

2.1.4. Robots bípedos y robots humanoides

Un robot bípedo es aquel robot caminante cuya locomoción se efectúa mediante dos únicas extremidades denominadas piernas (Figura 7). Con esta definición, claramente el ser humano es el modelo por excelencia de locomoción bípeda. Sin embargo no es el único ser vivo que utiliza la caminata bípeda como medio de locomoción. Los canguros y otros marsupiales bípedos utilizan sus piernas para saltar, ayudados de sus colas para estabilizarse. Las aves también son animales bípedos que utilizan sus piernas para saltar o caminar. No obstante utilizan otro medio de locomoción principal como volar o nadar, en el caso de los pingüinos [7].

Sin embargo, los humanos junto con los grandes simios son los únicos seres vivos que evolucionaron para usar la caminata bípeda como principal medio de locomoción. De ahí que se considere la forma de caminar del ser humano como principal fuente de inspiración en cuanto al estudio de la caminata bípeda. En este caso se requiere de especial atención en el control de su estabilidad postural, claramente mucho más difícil de asegurar que en el caso de los robots con cuatro o más patas. Es necesario recalcar que, aun cuando la complejidad de

la caminata bípeda es grande, la estructura bípeda es la forma de locomoción más flexible en términos de la evasión de obstáculos y del rápido reestructuramiento de caminado [6].

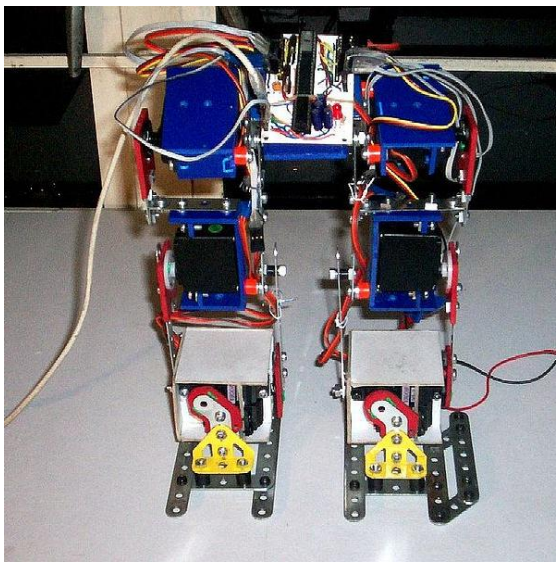


Figura 7. Robot bípedo

Figura 8. Robot humanoide

Existen dos puntos de vista para definir el término de un robot humanoide [7]:

a. Un robot se dice que es humanoide si la estructura cinemática de su cuerpo es similar a la de un ser humano, con lo cual posee cuerpo, cabeza, extremidades superiores y extremidades inferiores (Figura 8).

b. Un robot es humanoide si este actúa como ser humano. Esto es, si es capaz de realizar tareas normalmente efectuadas en un ambiente típico de humanos. Esta definición se basa en la funcionalidad más no en la apariencia y/o estructura del robot.

El segundo punto de vista implica que un robot humanoide debe contemplar alguna funcionalidad típica de un humano. Y una de estas funcionalidades puede ser la locomoción bípeda. De tal forma, un robot bípedo puede considerarse como un robot humanoide, pudiendo utilizar en algunas ocasiones los conceptos de robot humanoide y robot bípedo como sinónimos, pero un robot humanoide no es forzosamente un robot bípedo.

2.2. Inteligencia Artificial

Hasta el momento en este capítulo hemos descrito algunos conceptos teóricos importantes sobre la capacidad de robot y los tipos que existen. Sin embargo, un robot debe contar con cierto nivel de inteligencia para poder llevar a cabo sus funciones.

No es sencillo dar una definición del concepto “inteligencia”, pero podríamos describirla de forma cualitativa mediante la figura 9.

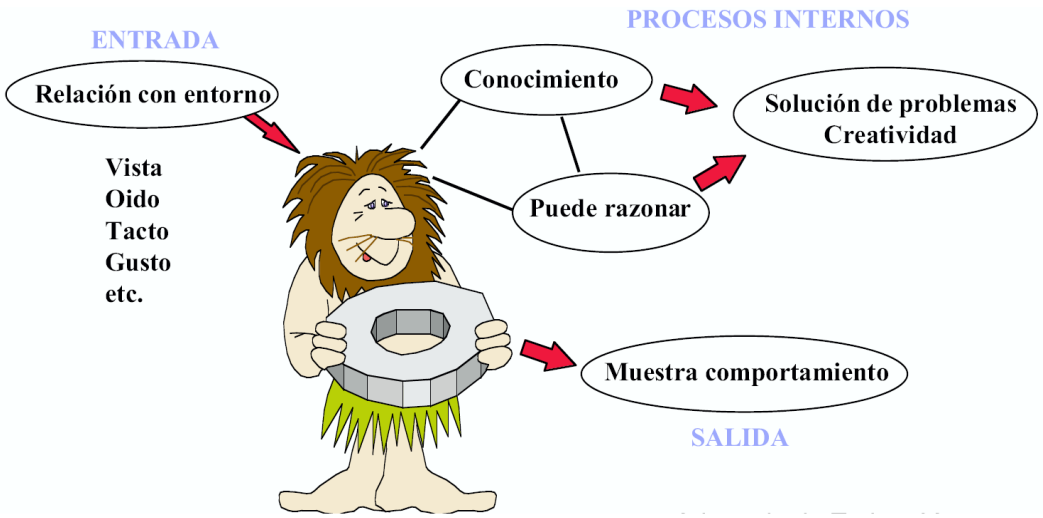


Figura 9. Los componentes de la inteligencia: entrada, procesos internos y salida

En general la inteligencia se puede describir como "la aptitud que nos permite recoger información de nuestro interior y del mundo que nos circunda, con el objetivo de emitir la respuesta más adecuada a las demandas que nos plantea la vida cotidiana. Primeramente se requieren datos de entrada y esta información la recibimos con nuestros sentidos (vista, oido, etc.) con lo cual hacemos un proceso interno para asimilar lo que percibimos (es decir razonamos y finalmente podemos generar una salida: comportamiento).

Así como nosotros contamos con esta inteligencia, tenemos que desarrollar una inteligencia para el robot. A esta inteligencia se le conoce como inteligencia artificial. En la literatura se conocen distintas definiciones de la Inteligencia Artificial. Kurzweill (1990) la define como el arte de crear máquinas que desarrollan funciones que requieren inteligencia de tipo humano. Shalkoff (1990) dice que es el campo de estudio que busca explicar y emular el comportamiento inteligente en

términos de proceso computacional. Winston (1992) menciona que es el estudio de la computación que hace posible la percepción, razón y acto [19].

Otra definición argumenta que la inteligencia artificial es una rama de las ciencias computacionales dedicada al desarrollo de agentes racionales no vivos [20]. Para explicar la definición anterior, entiéndase a un agente como cualquier cosa capaz de percibir su entorno (recibir entradas) a través de sensores, procesar tales percepciones y actuar en su entorno (proporcionar salidas) [19]. Por otro lado la racionalidad puede entenderse como la característica que posee una elección de ser correcta, más específicamente, de tender a maximizar un resultado esperado (este concepto de racionalidad es más general y por ello más adecuado que inteligencia para definir la naturaleza del objetivo de esta disciplina).

En general podríamos decir que inteligencia artificial es la disciplina que se encarga de construir procesos que al ser ejecutados sobre una arquitectura física producen acciones o resultados que maximizan una medida de rendimiento determinada, basándose en la secuencia de entradas percibidas y en el conocimiento almacenado en tal arquitectura [20].

El tipo de inteligencia que se utiliza en la simulación del robot es autómata finito y se tiene que saber lo indispensable con respecto a dicha inteligencia.

2.2.1. Autómata finito

Un autómata finito o máquina de estados finitos es un modelo matemático que realiza cómputos en forma automática sobre una entrada para producir una salida [21].

Las máquinas de estado finito son una herramienta muy útil para especificar aspectos relacionados con tiempo real, dominios reactivos o autónomos, computación reactiva, protocolos, circuitos, arquitecturas de software, etc.

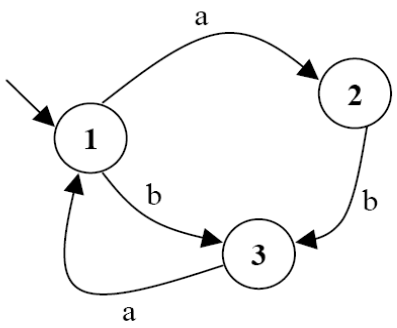
El modelo de FSM (*Finite State Machine*) es un modelo que posee sintaxis y semántica formales y que sirve para representar aspectos dinámicos que no se expresan en otros diagramas.

Sintaxis

Las máquinas de estado finito se definen como una tupla [22] $< S, \sum, A \subseteq S \times \sum \times S, sk >$, donde:

S = {s1, s2, …, sm}: es un conjunto finito de nodos.
□∑ : es un alfabeto finito de etiquetas.
A: es un conjunto finito de aristas etiquetadas que unen nodos.
sk ∈S : es el estado inicial.

Ejemplo



Donde:

- ⟨ *S* = {1,2,3}

- □∑ = {a,b}

- *A* = {(1, a, 2),(2,b,3) ,(3,a,1) ,(1,b,3)}
- *sk* =1 ⟩

Vale la pena destacar que formalmente una máquina de estado es la tupla anterior y no el “dibujo”. Este es tan sólo una representación gráfica de la máquina de estado para tener una más sencilla y rápida visualización de su contenido.

2.3 Juego de Fútbol entre Robots: Copa RoboCup

RoboCup™ [8] (originalmente llamado *Robot World Cup Initiative*) es una iniciativa internacional de investigación y educación. Esta iniciativa intenta fomentar la investigación en inteligencia artificial y robótica inteligente, proporcionando un problema estándar en donde se aplican diferentes tecnologías integradas dentro de distintos centros de investigación.

Con este fin, RoboCup eligió utilizar el juego de fútbol dentro de un camponato y conferencia denominado "*RoboCup World Championship and Conference"* (originalmente llamado *"The Robot World Cup Soccer Games and Conferences"*). El objetivo consiste en que un equipo de robots juegue un partido de fútbol, aplicando diversas tecnologías computacionales tales como principios de inteligencia artificial, diseño de agentes autónomos, colaboración multi-agente, adquisición de estrategias, razonamiento en tiempo real, robótica, y fusión de sensores.

RoboCup es una tarea para un equipo de múltiples robots de rápido movimiento en un ambiente dinámico. También ofrece una plataforma de software para la investigación sobre los aspectos del software de la RoboCup. Mientras el partido de fútbol es usado como un estándar de un problema en el que se amplían los esfuerzos concentrados e integrados, la competencia es sólo una parte de la actividad de RoboCup.

La intención de RoboCup es promover la investigación en inteligencia artificial y robótica al utilizar como dominio de prueba un juego mundialmente conocido. Una de la formas más efectivas de promover la investigación en ingeniería, además de los desarrollos orientados a una aplicación específica, es la de establecer una meta a largo plazo. Tal vez pensemos que lograr que un grupo de robots jueguen fútbol no representa un gran impacto social o económico, sin embargo conseguir esta meta ciertamente será un logro importante en el campo de la investigación científica.

El objetivo de la RoboCup es que en el año 2050, un equipo de robots humanoides compita contra el equipo campeón de la Copa Mundial de fútbol, según las reglas de FIFA [9].

Las actividades que conforman la iniciativa de RoboCup son:

- Conferencias técnicas
- Competencia mundial

- Competencias solucionando retos
- Programas educativos
- Desarrollo de Infraestructura

La *RoboCup World Championship and Conferences* es el pilar central de esta actividad, en donde los investigadores se reúnen anualmente para evaluar el progreso de la investigación en esta área.

En la actualidad, RoboCup tiene cuatro dominios principales:

- RoboCupSoccer
- RoboCupRescue
- RoboCup@Home
- RoboCupJunior

De estos cuatro dominios, en esta tesis nos concentraremos en RoboCupSoccer ya que en este dominio esta la categoría en la cual se pretende participar.

RoboCupSoccer. El principal enfoque de las actividades de RoboCup es la competencia de fútbol. Sin embargo cada juego también representa una oportunidad importante para la búsqueda de intercambio de información de la investigación realizada. También sirven como una gran oportunidad para la educación y el entretenimiento del público.

RoboCupSoccer esta dividió en las siguientes ligas (Figura 10):

- **Liga de humanoide**. Esta liga fue introducida en 2002. Robots humanoides bípedos autónomos juegan en partidos usando técnicas de reto. Esta liga tiene dos sub-categorías: *Kid-size y Teen-size*.

- **Liga de tamaño mediano (*Middle size*)**. Los robots de tamaño mediano no superan los 50 cm de altura; juegan fútbol en equipos de más de 6 jugadores con una pelota de color naranja; el campo tiene un tamaño de 12x18 metros. Los partidos están divididos en dos tiempos de 15 minutos cada uno. Se pueden utilizar cualquier sensor y los robots pueden comunicarse utilizando redes inalámbricas.

- **Liga de tamaño pequeño (*Small size*)**. Los robots pequeños no superan los 18 centímetros de altura; juegan fútbol con una bola de golf de color

naranja en equipos de más de cinco robots en un terreno de juego de tamaño de 6.5x4.5 metros. Los partidos tienen dos tiempos de 10 minutos cada uno. El objetivo de esta liga es la cooperación simultánea con un sistema centralizado/distribuido híbrido.

- **Liga de plataforma estándar (*Standard platform*)**. Esta liga reemplaza a la exitosa liga de cuatro piernas. Todos los equipos usan robots idénticos. Los equipos se concentran en el desarrollo del software solamente, mientras usan técnicas de comportamiento en los robots. Los robots operan completamente autónomos (p.ej. no hay control externo, ni por humanos ni por computadoras).

- **Liga de simulación**. Los jugadores se mueven en un campo virtual dentro de una computadora. Esta una de las categorías más viejas en RoboCupSoccer. Se dividen en dos subligas: 2D, 3D, desarrollo 3D y realidad mezclada.

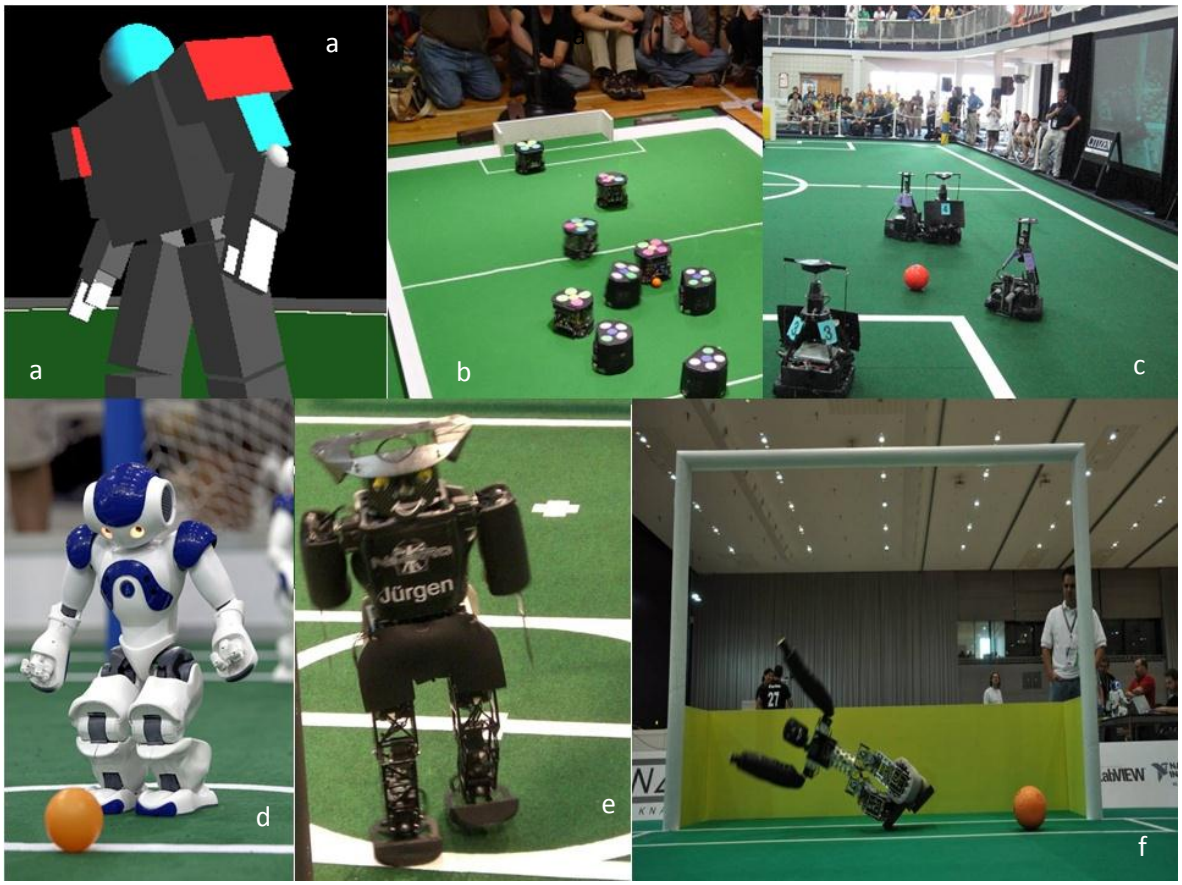


Figura 10.Categorias de RoboCup Soccer: a) liga de simulación, b) robots de tamaño pequeño, c) robots de tamaño medio, d) robots de plataforma estándar, e) kid size y f) teen size.

2.3.1. Robots móviles y robots humanoides.

En las ligas de robots físicos se pueden separar en dos tipos primordialmente: los móviles con ruedas y los humanoides.

1. **Los móviles con ruedas**. Son más fáciles de desarrollar ya que su medio de desplazamiento son las ruedas. En los robots de este tipo no se utiliza la cinemática y dinámica para su locomoción. Las ligas de competencia que utilizan este medio son:

- **Liga de tamaño pequeño**. Estos robots de fútbol toman lugar en equipos de cinco robots cada uno. El robot debe de tener las siguientes dimensiones de acuerdo a las reglas F180: el robot debe tener un diámetro de 180 mm. y no debe ser más alto de 15 centímetros. Los robot deben jugar en un campo de color verde con una longitud de largo de 4.9 metros por 3.4 metros de ancho con una pelota de golf naranja. Los robots pueden utilizar sensores de visión colocados directamente en el robot y utilizar visión global. Existe también la posibilidad de usar otro tipo de visión mediante cámaras que se colocan fuera del terreno del juego para poder visualizar todo el terreno de juego y pasarle los movimientos al robot para que se desplacen en el campo.

  En esta liga cada equipo utiliza una PC fuera del campo para comunicar los comandos del árbitro a los robots dentro del terreno. La comunicación es vía inalámbrica y típicamente usa señales FM para transmitir y recibir a los robots. La estrategia de juego está basada normalmente, que uno o dos jugadores defiendan, mientras que tres o cuatro ataquen.

- **Liga de tamaño mediano**. Al igual que las anteriores, en esta liga los robots se desplazan por medio de ruedas y cada equipo cuenta con cinco robots para el juego. El robot debe de tener una altura máxima de 50 centímetros, 50 centímetros de ancho y 80 centímetros de largo. El máximo peso del robot es de 40 kilogramos y la base del robot debe ser de color negro. La pintura debe ser en mate para disminuir el reflejo de la luz.

  Los robots de ambos equipos deben tener todos los sensores que utilizarán en el terreno de juego sobre ello mismos (ya no se utiliza la ayuda externa de por ejemplo cámaras como es el caso de liga pequeña). Está permitida la comunicación entre robots vía inalámbrica pero no está permitida la intervención de los usuarios directamente sobre el robot, excepto para colocar y quitar los robots del terreno de juego. No está permitido el uso de visión global, lo cual en muchos casos causa problemas de percepción para localizar a sus otros compañeros y a la pelota.

2. **Los robots humanoides**. Su locomoción es más difícil de desarrollar, ya que se debe tener un cierto control sobre la cinemática y dinámica del robot. Las áreas de los robots humanoides son:

- **Liga humanoide**. En esta liga, los robots son autónomos y tienen la estructura física de un humano. En cuanto a los sensores se exige que se cuente con un máximo de dos cámaras para poder hacer visión y que su campo de visión sea hacia al frente y que no supere los 120° de giro del cuello del robot. Los robots en esta categoría están divididos en dos clases: KidSize (de 30 centímetros a 60 centímetros de altura) y TeenSize (de 100 a 160 centímetros de altura). En esta liga los distintos campos de investigación que se aplican para mejorar el juego son: la caminata dinámica, correr y patear el balón manteniendo el balance, percepción visual de la pelota, reconocimiento a otros jugadores y al campo, la auto-localización y el trabajo en equipo. La mayoría de los mejores robots humanoides está en esta liga.

- **Liga de plataforma estándar**. En esta liga, todos los equipos utilizan el mismo tipo de robot. Los equipos solamente se concentran en el desarrollo del software y normalmente usan un autómata de estados finitos para la inteligencia del robot. Los robots operan completamente autónomos, p. ej. sin control externo por humanos o computadoras.

  Durante varios años el robot AIBO de la SONY (Figura 11) era el robot que se utilizaba en esta categoría y utilizaban cuatro patas como medio de locomoción en la RoboCup, sin embargo en el 2008 el robot Nao reemplazó al AIBO en esta categoría, por lo que se tiene que conocer más a fondo sobre este robot y se desarrolló un sub-capítulo acerca de este robot.

Figura 11. Robot AIBO de SONY utilizado para fútbol en cuatro patas

2.3.1.1 Nao

El robot Nao es el “robot estrella” de este proyecto, por eso se tiene que conocer lo básico acerca de él. Nao [11] es un robot humanoide autónomo, programable y de mediana estatura. Fue creado por la compañía Francesa Aldebaran Robotics. El proyecto Nao fue lanzado en el 2005 y de acuerdo a su creador Bruno Masinnier, estará disponible al público en 2011.

El robot Nao es el robot oficial para la RoboCup desde el 2007 reemplazando al AIBO y también es usado para el torneo del RobotStadium. Este robot ha evolucionad desde la versión 1, la cual contaba solo con una cámara, hasta la versión 2 la cual cuenta con dos cámaras para poder ubicar la pelota. Actualmente ya se cuenta con la versión 3 del Nao la cual se usó por primera vez en el torneo de RoboCup que se celebró en Singapur en Junio de 2010. Esta versión utiliza 25 grados de libertad y puede agarrar objetos con sus manos. En este proyecto de tesis se utilizó el Nao Versión 3.0 (Figura 12).

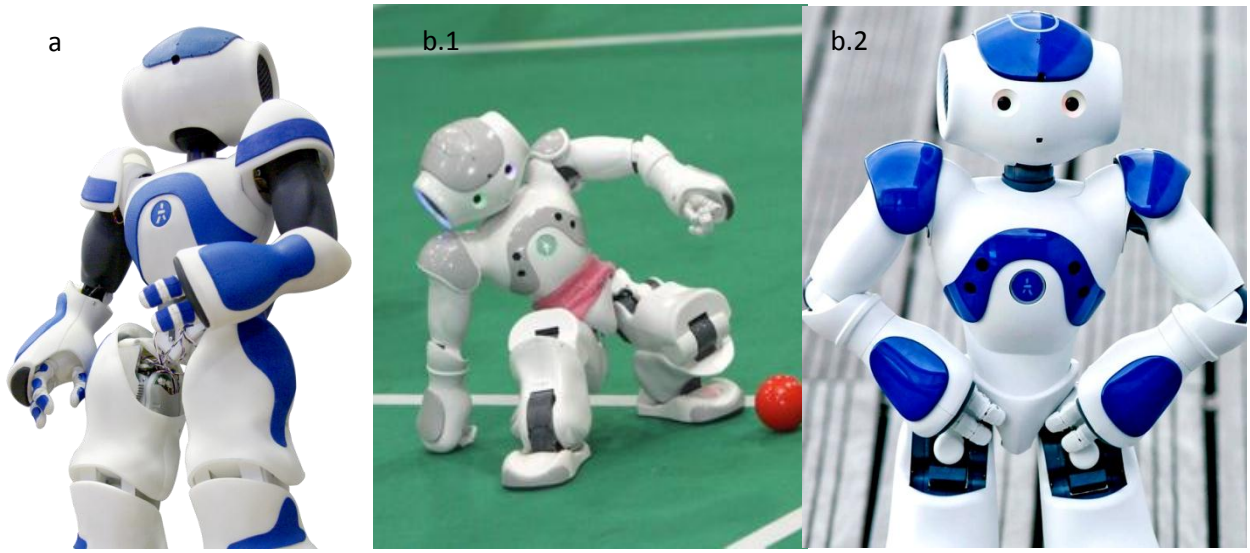


Figura 12. Distintos tipos de Nao: a) Nao V1, b.1) Nao V3 gris, b.2) Nao V3 Azul

Nao cuenta con las siguientes características [13]:

- 58 centímetros de altura.
- 4.3 kilogramos.
- 25 grados de libertad (Figura 12).

- Sensores de inercia (un acelerómetro y un giroscopio).
- 2 pares de sensores de ultrasonido.
- 4 micrófonos, reconocimiento de voz y análisis de sistema.
- 2 bocinas Hi-Fi.
- 2 cámaras con resolución de 640 x 480.
- 3 secciones de sensores de capacidad.
- Infrarrojo para envió/recepción de datos.
- Conexión WiFi (IEEE 802.11g).
- Sensores de presión.
- 2 *bumpers*.

Lo más interesante en cuanto a investigación se refiere es que Nao es completamente programable mediante herramientas como Choregraphe, un programa de aplicación grafica de Aldebaran Robotics, o el framework Webots que permite trabajar con simulación y descargar el código directamente al robot [13].

Todas estas características hacen que hoy día Nao sea un éxito en varias universidades para las tareas de investigación en el campo de la robótica y la Inteligencia Artificial. Nao es un robot realmente eficiente para varios dominios, sin embargo el principal inconveniente, por el momento, es su batería que dura cerca de 90 minutos.

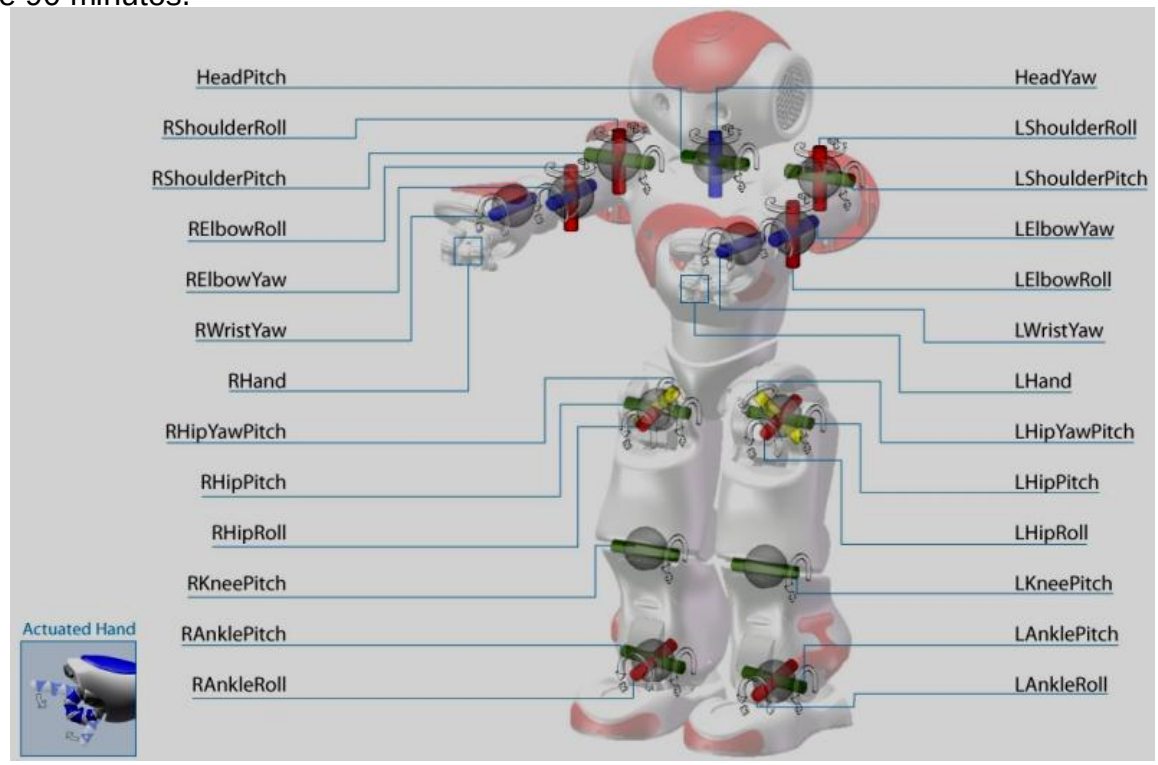


Figura 13. Grados de libertad del robot Nao

2.3.2. Robotstadium

Simultáneamente con el RoboCup, existe una competencia que se realiza todo el año, llamada RobotStadium [10], el cual toma lugar en línea en Internet y permite, a través del simulador Webots [17], implementar su propio código y de esta forma, hacer que los robots jueguen fútbol de manera simulada (Figura 14).

RobotStadium tiene el mérito de simular excelentemente la competencia de la RoboCup en la categoría plataforma estándar. Esta competencia da la oportunidad de programar en diferentes lenguajes; los robots simulados pueden ser controlados en C, C++, Java, Python o el lenguaje de script URBI.

RobotStadium organiza partidos de práctica entre los diferentes candidatos de todo el mundo. Tiene una clasificación llamada Pasillo de la fama, la cual almacena los mejores ocho equipos para poder ser parte de las finales durante la RoboCup. Esta es una herramienta práctica e interesante para aquellos que tratan de entrar en esa competición y no tengan la oportunidad de tener un robot real para poder practicar.

Los elementos del terreno de juego son: dos porterías, una de color azul y otra de color amarillo, tres robots por cada equipo y una pelota de color naranja.

Figura 14. Robots simulados en RobotStadium

2.4. Simulación del juego de fútbol mediante robots

En esta sección, se explicará el simulador Webots, los componentes de la interfaz y algunos requisitos importantes para su funcionamiento.

La simulación del software usado para jugar en RobotStadium es Webots [17]. Este es un simulador de realidad virtual para un robot profesional. Se desarrolló en el año de 1996 por Cyberbotics (empezó en EPFL, Suiza) y está disponible en los sistemas operativos de Windows, Linux y Mac OS [18].

El software de Webots permite al usuario crear mundos virtuales en 3D para la simulación de los robots y la creación de su ambiente, incluyendo las propiedades físicas como masa, distribución, fricción, inercia, etc. Los robots simulados pueden tener diferentes tipos de esquemas de locomoción, incluyendo los robots con ruedas, piernas, nadan o vuelan y pueden ser equipados con diferentes dispositivos de sensores y actuadores como los sensores de distancia IR, motor con ruedas, cámaras, servos, sensores de toque, giroscopios, emisores, receptores y muchos más [14].

Webots permite al usuario simular y modificar un gran número de robots existentes en el mercado (e-puck, AIBO de Sony, Nao, LEGO Mindstorm, Khepera, etc.) [15]. También es posible usar diferentes lenguajes de programación (URBI, C, C++, Java, Python and MATLAB) para poder entrenar a un Nao.

El motor de simulación de Webots puede realizar tanto la simulación de la cinemática como la simulación de la dinámica. Sin embargo tiene la libertad de elegir los aspectos físicos de la simulación (como son la fricción, el viento, etc). Las simulaciones cinemáticas no toman en cuenta las propiedades físicas de los elementos simulados (tales como peso, inercia o fricción) y además se ejecutan más rápido.

Para las simulaciones de interacciones físicas, Webots utiliza la librería ODE (*Open Dynamics Engine*). Esta es una librería muy poderosa para la simulación física. Webots es capaz de controlar la precisión en las interacciones físicas entre los robots y su medio ambiente. Además, Webots permite al usuario introducir librerías externas añadiéndolo a las fuerzas ODE y obtener formas que no están simuladas en un mundo estándar, como son el modelado de fuerzas hidrodinámicas, viento, etc.

La configuración mínima recomendada para poder utilizar Webots es [15]:

- Windows XP / Vista, Mac OS X 10.4 / 10.5 o Linux (2.6 kernel)
- 1 GHz de procesador
- 512 MB RAM / 300 MB espacio en disco
- Tarjeta gráfica NVIDIA o ATI con128 MB RAM

Webots ofrece la posibilidad de tomar imágenes PNG y guardar simulaciones como MPEG (Mac/Linux) y AVI (Windows). También es posible importar y exportar mundos Webots u objetos en el formato ***VRML*** (*Virtual Reality Modeling Language*).

2.4.1 Actividades con Webots

Webots es usado para proyectos de búsqueda y educacionales para robots móviles. Muchos proyectos móviles han recaído en las siguientes áreas:

- Prototipos de robots móviles (investigación académica, industria automotriz, aeronáutica, industria de aspiradoras, industria de juguetes, etc.)
- Investigación de la locomoción de robot (humanoides, con piernas, cuadrúpedos, etc.)
- Investigación multi-agente (inteligencia múltiple, robots móviles colaborativos, etc.)
- Investigación de comportamiento adaptado (evolución genética, redes neuronales, aprendizaje maquina, Inteligencia Artificial, etc.)
- Entrenamiento a robots móviles (programas en C/C++/Java/Python, torneos de robots, etc.)

2.4.2 Interfaz de Webots

La interfaz gráfica de Webots está compuesta principalmente en cuatro partes (Figura 15):

- **Scene Tree**. La ventana ***'Scene Tree'*** permite acceder a la configuración de los objetos que están presentes en el entorno de simulación, estos incluyen el o los robots, la pelota, las porterías, etc. Por ejemplo, si se selecciona el objeto DEF BALL ball se puede modificar la posición original de la pelota. No se pueden modificar las propiedades porque es la pelota oficial con la que se juega en las competencias.

- **Text Editor.** La ventana ***'Text Editor'*** permite acceder al código del programa. En esta ventana se pueden abrir, crear, guardar, actualizar, limpiar, construir, buscar, reemplazar y compilar.

- **Console.** La ventana ***´Console´*** permite desplegar resultados que se están ejecutando en el robot para tener una idea de los valores que tiene al interactuar con el medio ambiente y poder utilizar de una mejor manera los actuadores.

- **3D window.** La ventana ***´3D window´*** muestra los componentes que se establecieron en el *Scene Tree* de una forma visual, y cuando el código finalmente es ejecutado poder "observar" lo que hace el robot en el campo"

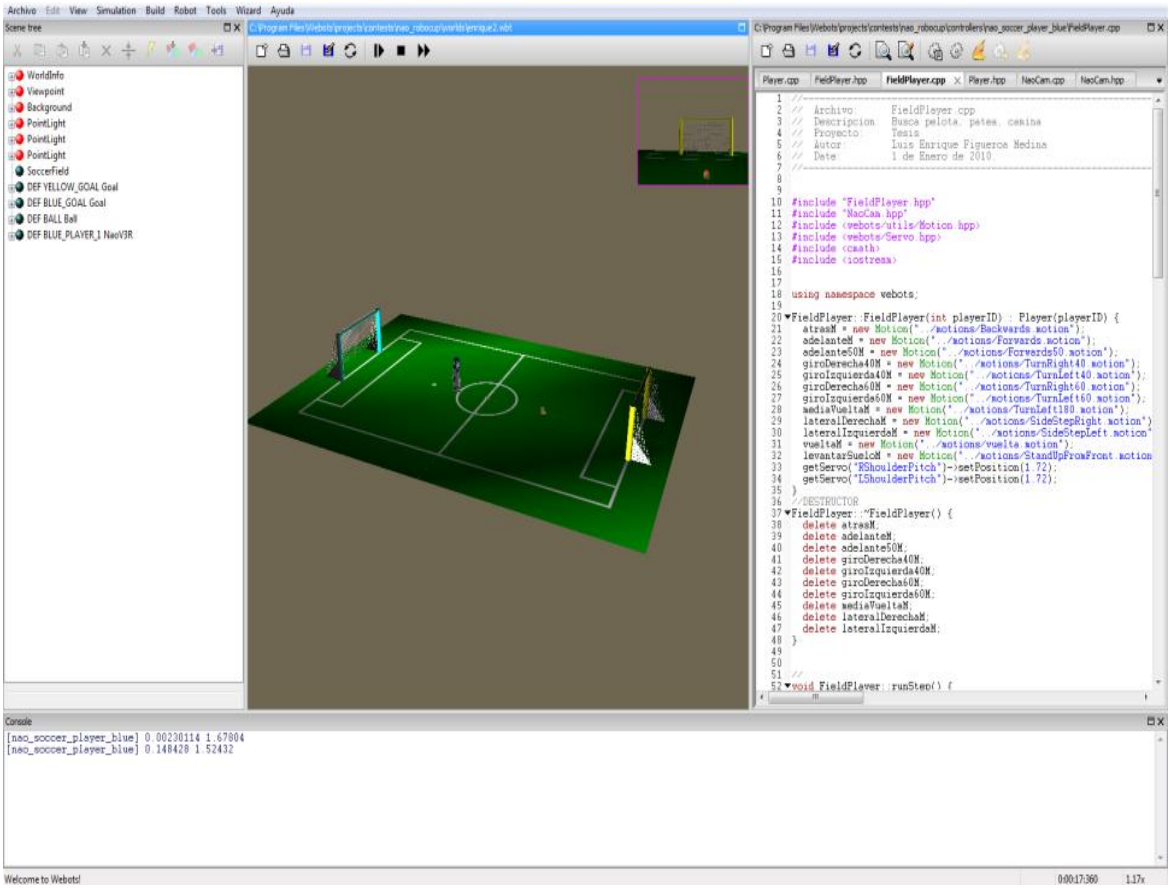


Figura 15. Interfaz gráfica del simulador Webots

2.4.3 Transferencia a robots reales

Una vez probados en el simulador los controladores del robot, estos pueden ser transferidos a los robots reales. En este caso para poder utilizarlos en el NAO, es necesario usar la herramienta URBI™ para Webots (Gostai SAS) o NaoQi™ y "nao_in_webots" (Aldebaran Robotics SA), para que finalmente se pueda escribir los controladores del programa y conectarlo a la simulación o al robot real Nao.

Capítulo 3

Simuladores y robots utilizados en RoboCup

3.1. Simuladores

En la actualidad existen diferentes tipos de simuladores comerciales para emular a robots bípedos y cuadrúpedos. Los siguientes siete simuladores son los más utilizados (sin incluir a Webots):

1. **UCHILSIM** [23]. Es un simulador robótico especialmente desarrollado para la liga de cuadrúpedos de RoboCup; fue el primer simulador que se utilizó para esta competencia .Reproduce con mucha exactitud la locomoción del AIBO y sus interacciones con los objetos en el terreno de juego.

   Las representaciones gráficas en el terreno de juego poseen un alto nivel de detalle y utilizan Open Graphic Lybrary (OpenGL) [24]. La meta principal del simulador es volverse una plataforma para aprender los comportamientos complejos robóticos, los cuales pueden ser directamente transferidos a los robots reales. UCHILSIM es capaz de adaptar sus parámetros automáticamente, con el comportamiento del robot real. El uso de este simulador permitió acelerar el progreso en la liga de los cuadrúpedos permitiendo a más gente participar en este reto.

2. **SimRobot** [25]. Este simulador no está limitado a una clase en especial de robots móviles. El lenguaje utilizado en este simulador está basado en XML, en donde los usuarios son libres de especificar cualquier robot y sus ambientes completamente sin la necesidad de otros lenguajes de programación.

   Diferentes partes del cuerpo de un robot, actuadores y múltiples sensores permiten la composición libre de un robot. Para simular la dinámica de cuerpos rígidos, usa Open Dynamics Engine [26] (ODE) y la visualización de las imágenes está basada en OpenGL.

3. **Übersim** [27]. Es un motor de simulación que soporta múltiples robots, para simular el juego de futbol entre robots. La meta de Übersim es crear un ambiente de simulación que permita controlar un rápido desarrollo y transferir el comportamiento a robots reales.

Übersim ha sido construido con especial atención en el diseño la simulación física, ya que reproduce con una alta fidelidad los modelos de fricción entre los objetos, colisiones entre los objetos e infinita aceleración de objetos tales como masa e inercia rotacional.

4. ***USARSim*** (*Urban Search and Rescue Simulator*) [28]. Es una simulación de alta fidelidad de robots en búsqueda y rescate urbano y ambientes, destinada como una herramienta de investigación para el estudio en la interacción humano-robot y la coordinación entre múltiples robots. El simulador fue diseñado como una extensión del juego del motor *Unreal Engine*, una plataforma de software comercial para el desarrollo orientado a acción en primera persona desarrollado por *Epic Games*.

   Dejando los aspectos más difíciles de la simulación, es una plataforma comercial que ofrece una representación visual superior y un buen modelado físico. Todos los esfuerzos en el desarrollo se centran en tareas específicas de la robótica, modelado de entornos, sensores, sistemas de control e interfaz de los instrumentos.

5. ***Gazebo*** [29]. Es un simulador para múltiples robots en ambientes externos. Es capaz de simular una población de robots, sensores y objetos en un mundo tridimensional. Genera sensores realistas e interacciones físicas entre objetos. Gazebo puede ser controlado a través de una interface de bajo nivel (*libgazeb*o). Esta librería permite a los desarrolladores integrar fácilmente Gazebo en sus propios robots o arquitecturas.

   Gazebo es usado normalmente en conjunto con un dispositivo llamado Player. Player provee un mecanismo (servo) a través del cual los controladores del robot (clientes) pueden interactuar con los robots reales y sensores. Player es de código abierto y software libre, liberado bajo la plataforma GNU.

6. ***Microsoft Robotics Studio*** [30]. Es un entorno basado en Windows para el control robótico y simulación. La simulación realística está provista por el motor PhysX de AGEIA, el cual posibilita la emulación por software y la aceleración por hardware.

   La aplicación es una multi-plataforma robótica en donde se permiten varios lenguajes como Visual C#, Visual Basic .NET, JScript, IronPython y lenguajes de terceras partes que se adecuen a la arquitectura basada en servicios. El desarrollador puede acceder fácilmente a los sensores y actuadores de los robots, el simulador proporciona una librería de implementación de concurrencia basada en .NET. La comunicación está basada en mensajes, permitiendo la comunicación entre módulos.

7. **SimSpark** [31]. Es un sistema genérico de simulación para diferentes simulaciones en paralelo. Este sistema apoya el desarrollo de simulaciones físicas y la investigación robótica con un marco de trabajo con código abierto para las aplicaciones.

   SimSpark era el motor de simulación para la RoboCup 3D de simulación de futbol en la liga (antes que lo reemplazara Webots). El modelo de fútbol robot original era conocido como soccerbot, pero en el 2008 se ha sustituido por el robot Nao. SimSpark incluye una visualización 3D basada en OpenGL. Soporta escenas simuladas y visualización en aplicaciones de monitor externo. Hay implementaciones para sistemas operativos Linux, Windows y MacOS X. SimSpark ofrece una interfaz grafica de usuario experimental, la cual permite reemplazar el modo consola.

3.2 Robots comerciales utilizados en Robocup

3.2.1 Robonova

Robonova (ver figura 16) es un robot de alta tecnología que contiene servos digitales de gran potencia y precisión y un sistema microcontrolador de última generación capaz de manejar hasta 40 dispositivos entre servos, sensores y otros controladores. Este robot posee 16 grados de libertad (5 en cada pierna y 3 en cada brazo).

Figura 16. Robonova de la compañía Hi-Tech

Con este robot se empezaron los primeros proyectos sobre robots humanoides en el departamento de Control Automático del CINVESTAV-DF. Los objetivos consisten en generar proyectos de investigación sobre áreas como el modelado, simulación, percepción, programación y control de este tipo de máquinas, así como desarrollar aplicaciones como el control basado en sensores exteroceptivos (fuerza, proximetría, visión e inerciales), el mapeo y localización simultáneos (SLAM), la navegación inercial, simulaciones, entre otros.

En la actualidad se sigue trabajando con el Robonova I, sin embargo están realizando el prototipo de su propio robot que saldrá a finales del año 2010 con el nombre de AH1N1 y con el trabajo de simulación del Nao para poder entrar a la competencia de RoboCup en la categoría Robotstadium.

3.2.2. AIBO

AIBO (***A****rtificial* ***I****ntelligence ro****BO****t*) [36] es un robot mascota fabricado por SONY lanzado a la venta en 1999. Tiene forma de perro y dispone de sensores que le evitan chocar contra objetos, y una cola que funciona como antena, además de "sentido del tacto". AIBO es uno de los juguetes más sofisticados que se pueden encontrar en el mercado, ya que usa una combinación de tecnologías robóticas y multimedia e inteligencia artificial para hacer posible que una serie de hardware y software interactúe entre sí.

AIBO se ha usado principalmente para la investigación de la inteligencia artificial, dado que integra una computadora, sistema de visión y motores de articulación. Los modelos más recientes son ERS-110 y ERS-111 y tienen un precio alrededor de los $2,500 US. Entre las funcionalidades del AIBO, caben destacar las más importantes:

- Aprendizaje.
- Reconocimiento de voz y cara.
- Autonomía.
- Programable.

Debido a las funcionalidades del AIBO de autonomía y programabilidad, la RoboCup realizó una competencia que tenía por nombre "Liga de Futbol para robots de Cuatro Piernas". En esta competencia sus competidores desarrollan sus programas para el AIBO y juegan entre ellos en las competencias. Esta liga tuvo lugar desde 1999 hasta 2008. Al final del 2008, las universidades de mayor renombre no participaron y se cambiaron a la plataforma Nao.

3.3. Nao en el mundo

Alrededor del mundo existen diferentes simulaciones del Nao utilizando Webots, cada uno de ellos cuenta con características diferentes. Cabe señalar que solo existe la idea de cómo desarrollaron cada uno: algunos mencionan algoritmos, otros autómatas y videos, pero ninguno explica en código cómo desarrollaron ciertas capacidades y muchos aspectos los dejan a la imaginación.

3.3.1. Nao- Humboldt en el equipo de Alemania

El equipo Nao-Humboldt [33] de la Universidad Humboldt de Berlín, Alemania, fue creado en el año 2007 y está compuesto por estudiantes e investigadores de esta universidad. Trabajan directamente con el robot en las actividades de visión y locomoción.

Las actividades de visión que realizó este equipo fueron las siguientes:

- Un sistema de visión que trabaja con imágenes YUV en una resolución de 160x120. La mayoría de sus algoritmos está basada en la clasificación de color como es el caso del algoritmo de espacio linear de color.

- Para la detección de la portería, usan un algoritmo de detección de portería que recae principalmente en el cálculo de medidas estadísticas para listas de pixeles de igual color.

- La detección de línea está hecha sin clasificación de color. Escanea la imagen completa en dirección horizontal y vertical buscando por líneas e indicando el posible comienzo y fin de una línea. Encontrado estos dos puntos, se calcula por un operador de Sobel.

- La detección de un robot usa áreas de color azul y rojo en la imagen. Estas marcas representan distintas partes de un robot (i.e. cabeza, hombros, etc.). Las partes del cuerpo identificados son tratadas para identificar a un robot. La posición y orientación de los robots detectados pueden ser fácilmente extraídas de las relaciones geométricas entre las áreas de color que lo conforman.

Las actividades de locomoción fueron las siguientes:

- Para movimientos como son el de patear o levantarse, usaron un método basado en fotogramas claves. Este método describe los movimientos a utilizar y los guardan en un archivo para ser llamados. Cada fotograma guarda la posición de los motores y los ángulos que se deben de rotar con una condición especial y duración.

- Para las demás locomociones como son caminar, se utiliza la cinemática inversa, ya que no se tiene que especificar cada ángulo y la posición de los motores. Esta locomoción puede ser alternada por la longitud del paso, la altura del paso y la velocidad. Estos movimientos son generados al instante.

3.3.2. Nao-EPFL en Suiza

En la universidad de EPFL (*Ecole Polytechnique Fédérale de Lausanne*) en Suiza, se está trabajando con la versión simulada para la categoría RobotStadium del torneo RoboCup [34]. Este equipo utiliza como base el simulador Webots. El trabajo de este equipo consiste en cuatro puntos importantes:

- **Moción para levantarse**. Esta locomoción fue realizada con la herramienta "Webots Motion Editor", con este apoyo se simplificó el trabajo. Por cada paso, cada actuador que el usuario quiere mover puede ser seleccionado e ir hasta donde quiera llevarlo. Cuando la secuencia de acciones está hecha, puede ser guardada en un archivo con la extensión .motion y puede ser ejecutado durante el juego usando la función "playMotion ()". Con esta herramienta, cada actuador no necesita ser llamado uno después del otro.

- **Optimización de las locomociones del robot**. Para la optimización de la locomoción del robot se utilizó el caminado más rápido en línea recta. Primero se utilizó el caminado original y después con el "Webots Motion Editor" algunas transiciones fueron cortadas y pegadas. Después de esto, el caminado fue más rápido que el original. En promedio con este método el caminado de 23 segundos lleva a caminar el robot por 2.37 metros. El segundo punto es el caminado en curva. En este caso se realizó un caminado con un ángulo de 15°, para lo cual la locomoción fue implementada utilizando el "Webots Motion Editor".

- **Detección de la portería**. En este punto, el equipo mejoró el algoritmo original de detección de la portería del simulador Webots. El código original detecta los pixeles de la portería con un algoritmo de detección de mancha.

- **Estrategia para disparo**. En este punto, el equipo implementó la siguiente estrategia. Cuando el robot está de frente a la pelota y con dirección a la portería, este camina con pasos pequeños hasta estar a 1.8 centímetros de la pelota. Cuando llega a esta distancia se prepara para patear la pelota con la pierna izquierda. El equipo modificó el algoritmo original de Webots para patear la pelota, ya que siempre se caía el robot después del disparo.

3.3.3. Nao-HTWK de Leipzig

El equipo Nao-HTWK pertenece a la universidad de Leipzig, Alemania. Este equipo fue creado en el 2009 para participar en la liga de plataforma estándar de la RoboCup. Entre sus principales desarrollos están la locomoción, la visión y la estrategia del juego.

En la locomoción aparte de caminar y patear, su principal logro fue el tiempo de levantado del robot boca abajo. Este tiempo es de 4.5 segundos, y es mucho más rápido que la locomoción que trae por *default* el Nao.

En el desarrollo de la visión, en este equipo se utiliza únicamente la cámara que se encuentra en la barbilla, porque el intercambio entre ambas cámaras toma demasiado tiempo utilizando la versión de NaoQI. Esto trae la ventaja de que la calibración de la cámara solo se tiene que realizar una vez. Utilizan el formato YUV para la composición de los colores en lugar del formato RGB. Utilizan un algoritmo para la detección de la pelota y otro algoritmo para la detección de las porterías.

Este equipo utiliza una estrategia basada en una máquina de estados finitos, en donde seis estados se encuentra en un ciclo (ver Figura 17). El ciclo consiste en los siguientes pasos y sus correspondientes mociones y procesos:

- **Encontrar pelota (*find ball*).** El robot se queda buscando la pelota en el terreno de juego y solamente puede cambiar de estado hasta que la pelota ha sido encontrada.

- **Girar hacia la pelota (*turn to ball*).** Dependiendo en que dirección se encuentre, el robot gira hacia la dirección de la pelota y cuando la tenga enfrente pasa al siguiente estado.

- **Caminar hacia la pelota (*walk to ball*)**. Al caminar hacia la pelota, calcula la distancia en que se encuentra su objetivo para que no se pase de la pelota.

- **Ajustar a la pelota (*adjust on ball*).** Cuando esta cerca de la pelota, el robot tiene que avanzar con pasos cortos para que no cometa el error de pegarle, cuando está colocado pasa al siguiente estado.

- **Girar alrededor de la pelota (*turn around the ball*)**. Si la portería no está enfrente, tiene que reacomodar su posición, por lo tanto gira en torno de la pelota para poder ubicar la portería enfrente del robot.

- **Disparar (*shoot*).** Finalmente la pelota está acomodada y la portería está enfrente del robot. El robot dispara y vuelve al estado 1 y repite de nuevo todo el proceso.

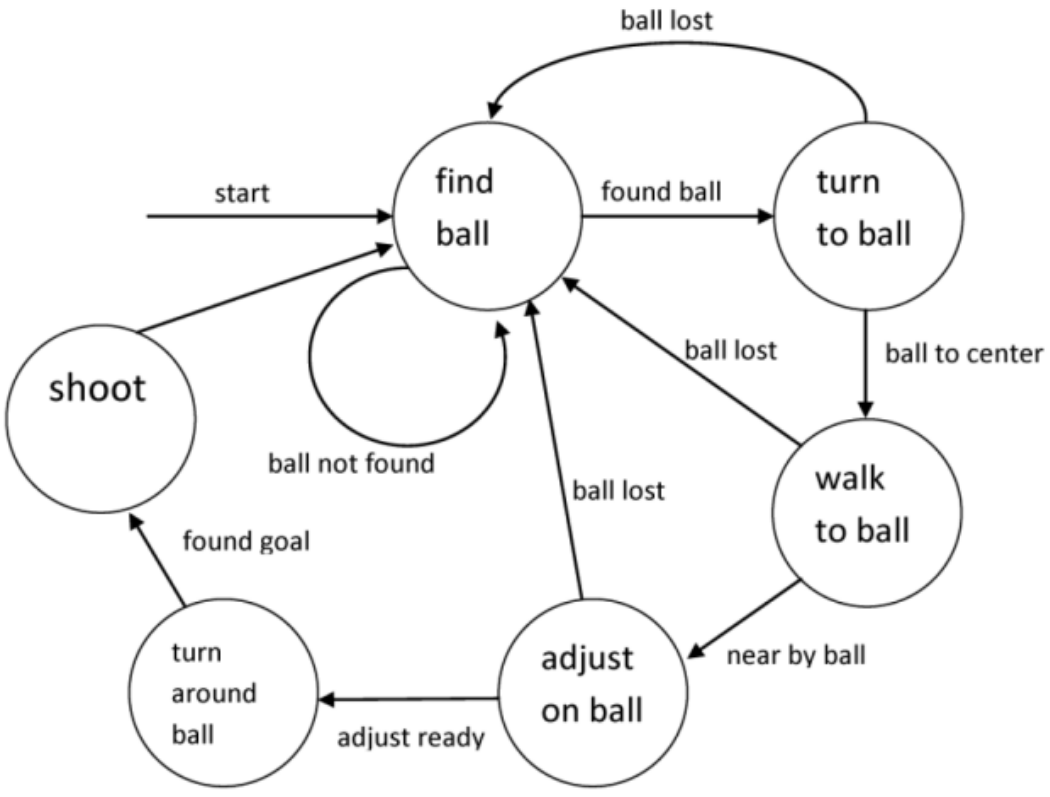


Figura 17. Maquina de estados finitos de Leipzig

3.3.4. Nao-Certo en Suiza

En el año 2008, en la universidad EPFL, Giuseppe Certo empezó a trabajar en el proyecto de simuladores para participar en la liga RobotStadium. Lo sobresaliente de este proyecto es la forma en que calculan la distancia y dirección de los objetos [13] (en este caso la pelota).

Para calcular la distancia de la pelota, primeramente se obtiene el ángulo *Alfa*, el cual se calcula a partir del ángulo que produce la cámara del robot (que se encuentra en su cabeza) mirando a la pelota. Después se obtiene la distancia *h* que hay entre el suelo y la cámara del robot (la cual es aproximadamente su tamaño). A partir de estos 2 datos, mediante un simple cálculo de trigonometría se obtiene la distancia del robot a la pelota. Para determinar el ángulo de dirección entre el robot y la pelota, es el mismo principio que para la distancia, excepto que esta vez el cálculo es mucho más simple. Ciertamente, para determinar la dirección de la pelota, es suficiente conocer la posición de la cabeza y a la vez la posición del servo *HeadYaw* con lo cual se obtiene el ángulo.

La estrategia que utilizan los robots en este equipo para el ataque es:

1. **Localizan la pelota**. Entra en un ciclo hasta que la pelota es encontrada y así pasa finalmente al punto 2.

2. **Pelota encontrada**. Si encontró la pelota, pasa al punto 3.

3. **Jugar**. El robot empieza a seguir la pelota, patearla, etc., hasta que el robot ya no ve más la pelota, pasa al punto 1 y se repite el ciclo.

La estrategia que utilizan sus robots para la defensa es:

1. **Localizar la pelota**. Si la pelota es localizada pasa al estado dos, de lo contrario se queda en este punto.

2. **Pelota encontrada**. Si la pelota está lejos de donde se encuentra pasa al estado tres, de lo contrario pasa al estado cuatro.

3. **Pelota lejos**. Se queda esperando a la pelota y cuando se encuentra cerca pasa al estado cuatro.

4. **Pelota cerca**. Se acerca a la pelota y la patea hacia el sentido opuesto y regresa al estado uno.

Capitulo 4

Simulación robótica

4.1 Simulador Webots

El programa comercial Webots cuenta con la siguiente estructura de diseño (ver figura 17) para poder desarrollar la simulación del Nao en este ambiente.

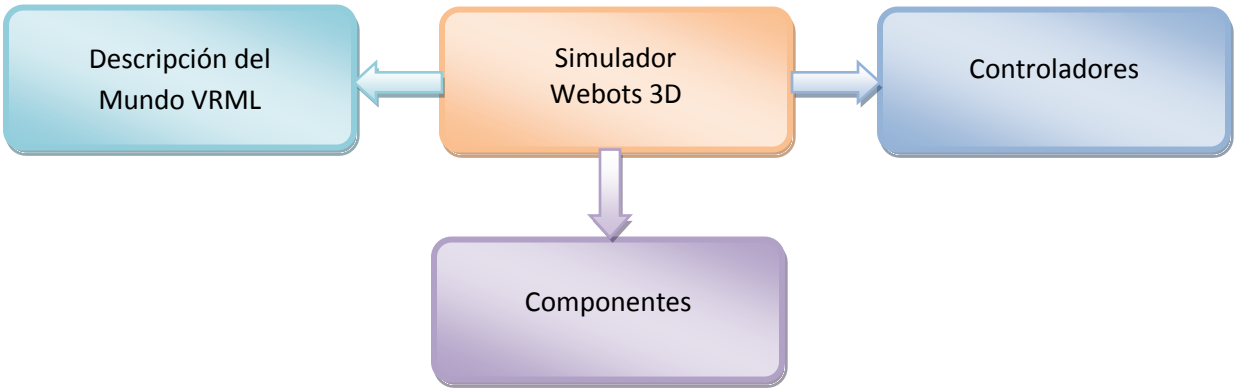


Figura 17. Estructura de diseño del simulador Webots

Como se observa en la figura anterior, el simulador Webots está integrado de tres partes:

- **Descripción del Mundo VRML**: un mundo en Webots es un ambiente virtual en 3D, el cual permite crear objetos y robots. Un mundo es almacenado en el directorio *Worlds* con la extensión *.wbt*. Este archivo contiene una descripción para cualquier objeto: su posición, orientación, geometría, apariencia (como color, brillo), propiedades físicas, tipos de objeto, etc. Un mundo es una estructura jerárquica donde los objetos pueden contener otros objetos (como en VRML97). Por ejemplo, un robot puede contener dos ruedas, un sensor de distancia y un servo que contiene una cámara. Esto hace que la cámara pueda rotar sobre su propio eje gracias al servo. Sin embargo, un archivo *world* no contiene toda la información necesaria para ejecutar la simulación. El controlador de cada

robot está especificado en el archivo *world*, el cual hace referencia a un archivo binario ejecutable (pero el archivo *world* no contiene ningún archivo binario ejecutable).

- **Componentes**: un componente es aquel elemento que pertenece al mundo virtual y es la parte "física" del mundo. Los componentes que se utilizan en la simulación son el robot, las porterías, la pelota, el terreno de campo y las luces.

- **Controladores**: un controlador es un archivo binario, el cual es usado para controlar a un robot que se ha descrito en un archivo *world*. Los controladores son almacenados en subdirectorios de Webots en el directorio *controllers*. Los controladores pueden ser de diferentes tipos:

    - Archivos ejecutables (.exe bajo el sistema operativo Windows)
    - Archivos binarios de java (.class)
    - Archivos de C y C++ (.c y .cpp), de Python (.py) entre otros.

  Una vez definidos en el campo los objetos y sus respectivas cuestiones físicas, solo falta manipular estos objetos. En este caso, el único objeto que se puede manipular es el robot, y este a su vez puede interactuar con los diferentes componentes que lo integran. Los controladores pueden ser escritos en diferentes lenguajes como se explicó en el capitulo 2. Cabe destacar que en este proyecto se utilizo el lenguaje C++ para realizar llamadas a objetos y así poder tener el programa escalable, es decir, si al robot se le agregara un sensor nuevo, solo sería crear el objeto y llamarlo y no tener que reestructurar todo lo que se tiene.

4.1.1. Mundo VRML

El VRML (*Virtual Reality Modeling Language*) o Lenguaje para Modelado de Realidad Virtual es un formato de archivo normalizado que tiene como objetivo la representación de escenas u objetos interactivos tridimensionales, diseñado particularmente para su empleo en la Web.

El lenguaje VRML posibilita la descripción de una escena compuesta por objetos 3D a partir de prototipos basados en formas geométricas básicas o de estructuras en las que se especifican los vértices y las aristas de cada polígono tridimensional y el color de su superficie.

Webots se basa principalmente en la sintaxis de VRML, para poder hacer referencia a la lista de nodos que se describen en la utilización de los objetos. Webots usa un subconjunto de nodos y campos de VRML97, pero también define nodos adicionales y campos específicos para las definiciones robóticas. Por ejemplo el código para diseñar una esfera en Webots usando VRML es:

```
Sphere {
  SFFloat   radius       1
  SFInt32   subdivision   1
}
```

La palabra “Sphere” representa el nombre del nodo que se va a utilizar. La esfera será colocada en el centro (0,0,0) de acuerdo al sistema de coordenadas local (ver figura 18). SFFloat y SFInt32 son la clase de datos que se utilizarán para la esfera. SFFloat es de tipo flotante para poder representar el radio del círculo, si el valor es positivo, sus caras de afuera de la esfera son mostradas, mientras si el valor es negativo, las caras de adentro son mostradas. SFInt32 se utiliza para describir el campo de subdivisión, el cual controla los números de caras que se utilizan en la esfera para representarla. Las esferas son representadas como icosaedros con 20 caras cuando el campo de subdivisión es cero. Si el campo de subdivisión es uno (el valor por defecto), entonces cada cara es subdividida en cuatro caras, haciendo 80 caras. Cuando una textura es aplicada a la esfera, la textura (*texture seem*) cubre la superficie por completo. La descripción del mundo VRML que se utilizó para esta tesis se encuentra en el apéndice.

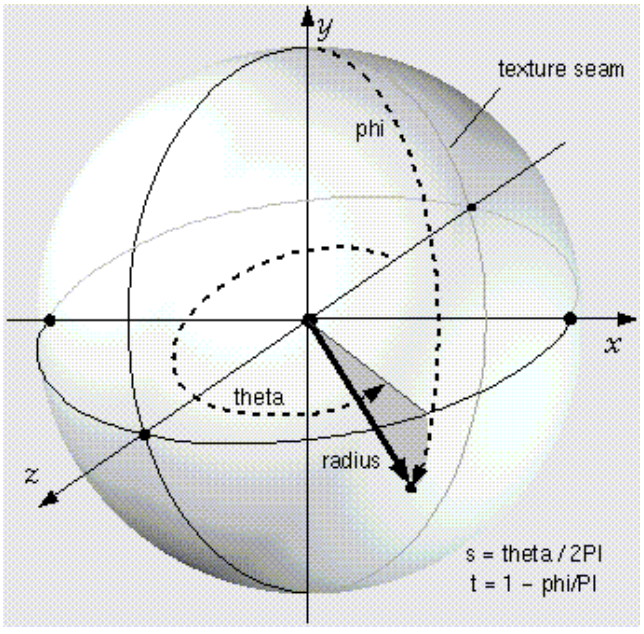


Figura 18. Características de una esfera en VRML.

La estructura jerárquica de un mundo virtual en VRML (ver figura 19), está compuesta por un Escenario de Grafos (*Scene Graph*), donde cada círculo representa un Nodo (*Node*) que posee cierta funcionalidad y los nodos padres agrupan a otros nodos hijos. Los nodos pueden ser de tipo forma (*Shape*) y poseen dos campos que son la geometría (*geometry*) y la apariencia (*appearance*), en donde la geometría trata sobre la forma (vértices, normales, etc.) mientras que la apariencia especifica su material, textura entre otras cosas. Al aprender de forma completa VRML ya se pueden crear otras figuras para el desarrollo de nuevos robots o elementos con los que puede interactuar.

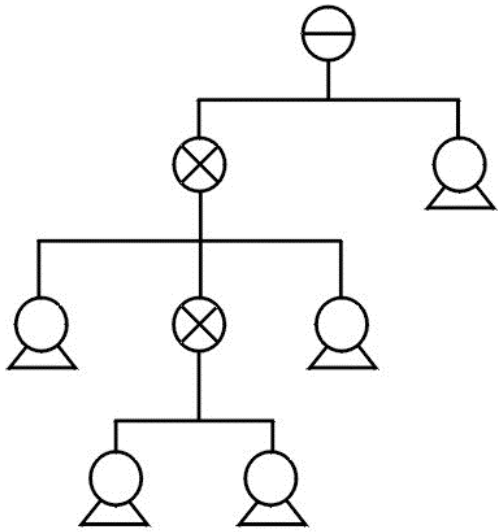


Figura 19. Estructura jerárquica de un mundo VRML.

4.1.2. Componentes de un robot Nao

Las componentes del b Nao, se pueden dividir principalmente en dos bloques:

- **Sensores**. El robot Nao está diseñado para poder percibir principalmente el ambiente que lo rodea. Para lograrlo cuenta con dos cámaras, micrófonos, altavoces, sensores de tacto, *bumpers* en los pies, giroscopio, acelerómetro y ultrasonido. Con estos sensores es capaz de “ver”, “escuchar”, “hablar” y “sentir”.

- **Actuadores**. Nao cuenta con 22 grados de libertad, lo que le permite desplazarse de un lugar a otro sin ningún problema. Si comparamos a Nao con la versión comercial de robot Robonova, este último solo cuenta con 16 grados de libertad y no puede girar sobre su propio eje. Para que el Robonova pueda girar sobre su propio eje, tiene que efectuarlo mediante

derrapes. A diferencia del Robonova, Nao realiza su giro sobre su eje, gracias al grado de libertad que tiene en sus pies y así puede efectuar este movimiento.

Los 22 grados de libertad se explican a continuación (ver figura 20):

- **Dos grados de libertad en la cabeza**. HeadYaw es el grado de libertad que permite a Nao tener un movimiento de izquierda a derecha con un rango en grados de 120 a -120. HeadPitch le da la posibilidad de mover la cabeza de arriba hacia abajo con un rango de -45° a 45°.

- **Cuatro grados de libertad en el brazo derecho y cuatro grados de libertad en el brazo izquierdo**. Para el brazo derecho se utilizan los siguientes grados de libertad: RShoulderRoll se ubica en el hombro derecho del Nao y permite mover el hombro de izquierda a derecha con grados de -95 a 0. RShoulderPitch permite mover el hombro derecho de arriba hacia abajo con grados de -120 a 120 respectivamente. RElbowRoll permite mover el codo derecho de izquierda a derecha con grados de 0 a 90. Finalmente RElbowYaw permite mover el codo derecho de arriba hacia abajo con grados de -120 a 120.

  Para el brazo izquierdo se utiliza LShoulderRoll, LShoulderPitch, LElbowRoll y LElbowYaw. Los valores que difieren son LShoulderRoll y LElbowRoll, ya que son valores espejos, es decir LShoulderRoll tiene valores de 0 a 95 y LElbowRoll va de -90 a 0.

- **Tres grados de libertad en la pierna derecha y tres grados de libertad en la pierna izquierda**. Para la pierna derecha se utilizan los siguientes grados de libertad. RKneePitch permite mover la rodilla derecha de arriba hacia abajo con grados de 0 a 130, respectivamente. RAnkleRoll permite mover el talón derecho de izquierda a derecha con grados de -25 a 45. RAnklePitch permite mover el talón derecho de arriba hacia abajo con grados de -75 a 45. Para la pierna izquierda se utiliza LKneePitch, LElbowRoll y LElbowPitch. Los valores que difieren son LElbowRoll, cuyo valor espejo tiene valores en gados de 25 a -45.

- **Dos grados de libertad en las caderas**. Los servos que hacen a Nao mover las caderas son RHipYawPitch en el lado derecho y en su contraparte LHipYawPitch en el lado izquierdo. Ambos servos tiene grados de -90 a 0.

Para mayor detalle de los valores de grados de los servos ver Tabla1.

Cabeza	HeadYaw	-120 a 120
	HeadPitch	-45 a 45
Brazo Izquierdo	LShoulderPitch	-120 a 120
	LShoulderRoll	0 a 95
	LElbowYaw	-120 a 120
	LElbowRoll	-90 a 0
Pierna Izquierda	LHipYawPitch	-90 a 0
	LKneePitch	0 a 130
	LAnklePitch	-75 a 45
	LAnkleRoll	-45 a 25
Pierna Derecha	RHipYawPitch	-90 a 0
	RKneePitch	0 a 130
	RAnklePitch	-75 a 45
	RAnkleRoll	-25 a 45
Brazo derecho	RShoulderPitch	-120 a 120
	RShoulderRoll	-95 a 0
	RElbowYaw	-120 a 120
	RElbowRoll	0 a 90

Tabla 1. Valores de cada articulación en grados

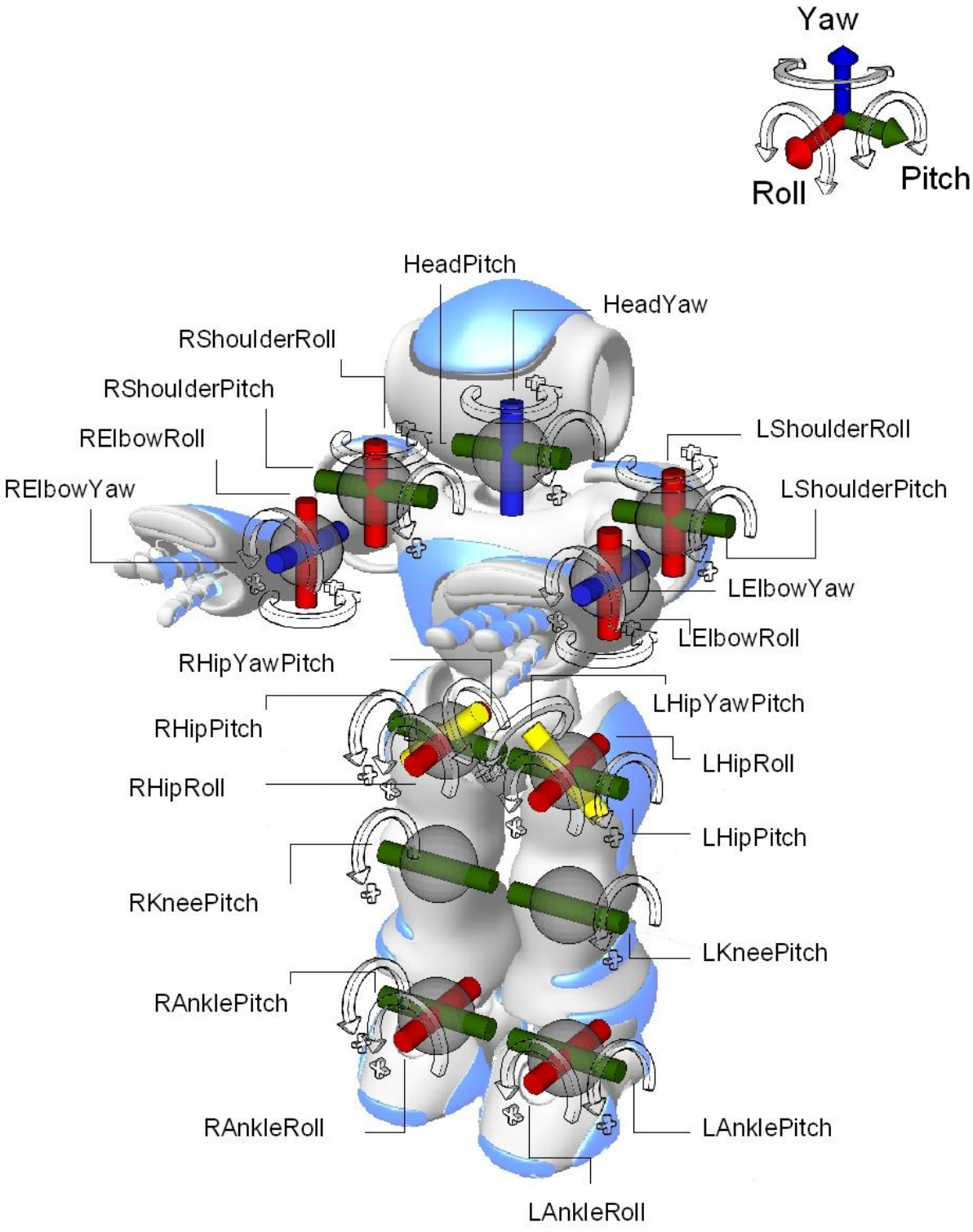


Figura 20. Representación de las direcciones y articulaciones del Nao

4.1.3. Controladores

En la simulación, durante un juego de futbol, seis jugadores robots están presentes en el campo, tres por cada equipo, dos atacantes y un portero respectivamente. Desde el punto de vista de la programación, se debe conocer que durante un partido, dos tipos de controladores se ejecutan al mismo tiempo pero con funcionalidades diferentes. El primer control es llamado Soccer Player y el segundo es denominado Supervisor [13].

- **Soccer Player**. Se refiere al programa que toma control en los robots. Este es el programa que cada participante deberá modificar para poder enfrentar a sus contrincantes. Solamente un Soccer Player es lanzado para cada robot de un equipo en el campo. Para su implementación, cada participante es libre de escoger el código en que desee programar y no existe ninguna restricción. Un programa de base es provisto en la instalación de Webots, y a partir de este código, cada participante lo modifica para tener robots más competitivos que los de los demás.

- **Supervisor**. Este programa toma el control del juego y debe acatar las reglas de la RoboCup. Este programa, a diferencia del Soccer Player, es el mismo para todos los participantes. Solamente un supervisor es lanzado por juego. El árbitro del partido se encarga de posicionar a los robots desde el principio de cada partido, también llevar el control de los goles anotados, así como el tiempo que resta al partido. También se encarga de reemplazar el balón cuando este sale del campo. Además, el Supervisor envía datos a todos los robots cada 500 ms (estos datos contienen el estado del juego, el marcador, el tiempo restante por jugar y otra información relevante al partido).

4.2. Utilizando Webots en Windows

La versión de Webots que se instaló fue la 6.1.5 (a la fecha de octubre de 2010 está la versión 6.3) porque esta versión es compatible con la llave Dongle (Figura 21) que se compró para este proyecto y funciona únicamente con esta versión y no es compatible con versiones posteriores.

Figura 21. Llave Dongle para utilizar Webots.

De la página oficial de Cyberbotics http://www.cyberbotics.com/archive/windows/ se puede bajar una versión de prueba que es funcional por 30 días y que permite ser utilizada de forma ilimitada para la simulación del robot Aldebaran Nao dentro del entorno de la Standard Platform de RoboCup. En esta página se busca webots-6.1.5_setup.exe que es la versión que se utilizó para empezar a desarrollar esta simulación (se recomienda utilizar esta versión, porque versiones posteriores modificaron al robot Nao físicamente).

Los requisitos mínimos para ejecutar este programa en una PC son:

PC con procesador de 1 GHz o más.
Tarjeta grafica de video de 128 MB (Tarjeta NVIDIA recomendada)
1 GB de disco duro disponible.

La infraestructura con la que se contó para el desarrollo de esta simulación es:

PC con Core 2 Duo a 2.0 GHz
Windows XP Service Pack 3
320 de Disco Duro
Tarjeta de Video NVIDIA 512 de tarjeta dedicada
Memoria RAM de 4 GB.

Para que todo funcione correctamente debemos tener instalada la tarjeta gráfica. Si no disponemos de este tipo de aceleración, en muchas ocasiones el robot no realizará correctamente lo que le indiquemos (por ejemplo, al decirle que avance en línea recta puede que se caiga de espaldas como sucedía con otra computadora). Esto es debido a que se necesita bastante tiempo de CPU para refrescar la imagen en el simulador y eso es tiempo de proceso perdido por el robot, ya que los procesos en el robot deben ejecutarse en tiempo real. En

cambio, si disponemos de aceleración gráfica, la mayoría del tiempo de CPU se dedica al robot Nao porque se delega a la tarjeta gráfica el refresco de la imagen.

La instalación de Webots en Windows es de forma “autoinstalación”. Para la instalación hay que ejecutar el programa el cual nos indicará en todo el camino los pasos para instalarlo. Una vez instalado el programa en nuestra computadora, lo ejecutamos desde el escritorio y nos aparece una ventana preguntándonos si queremos probar el programa.

En este momento ya tenemos instalado y en función por 30 días el programa Webots. Una vez instalado Webots, este nos provee de un ejemplo de simulación de juego de futbol entre dos robots, llamado Robostadium Contest. En esta simulación el robot Nao rojo solamente camina y el robot Nao azúl se dedica a observar y seguir la pelota.

Para poder utilizar la funcionalidad del robot azul, debemos eliminar al robot rojo. Al seleccionar el robot azul en Webots, podremos mover al robot dentro del campo. Nuestro primer paso es colocar el robot a la mitad del campo con la pelota enfrente y la portería amarilla enfrente del robot. Una vez hecho esto debemos salvar la posición y las características del nuevo mundo creado mediante Save World As, en el archivo Nao-Cinvestav.wbt.

Si se quiere integrar nuevos elementos al escenario, al finalizar se tiene que guardar en un nuevo mundo (si no se quiere perder el mundo anterior), porque al efectuar la ejecución, se perderán todos los elementos que se han agregado y volverá a la normalidad con el/los robot/s ya establecidos previamente.

4.3. Árbol de escenas en Webots

Al abrir el archivo de mundo Nao-Cinvestav.wbt, se pueden observar que ya existen nodos en nuestro árbol de escenas (*Scene tree*) que contiene cinco elementos físicos y seis elementos intangibles (ver figura 22).

Los nodos pueden ser expandidos con un doble click. Cuando un campo (*field*) es seleccionado, su valor puede ser editado en la parte inferior del árbol de la escena.

Los siguientes botones están disponibles para editar el mundo:

 **Cortar:** corta el nodo seleccionado.

 **Copiar:** copia el valor del nodo o campo seleccionado

Pegar: pega el valor del campo seleccionado o nodo.

Pegar después: pega un nodo debajo del nodo actual seleccionado.

Eliminar: Borra el nodo o campo seleccionado

Reset: el campo seleccionado le devuelve los valores predeterminados.

Transform: Permite cambiar el tipo de algún nodo.

Insertar después: inserta un nodo a la derecha después del nodo seleccionado

Nuevo nodo: agrega un nodo en el campo nodo.

Exportar: exporta un nodo en un archivo ASCII.

Importar: importa un nodo exportado previamente en el árbol de la escena.

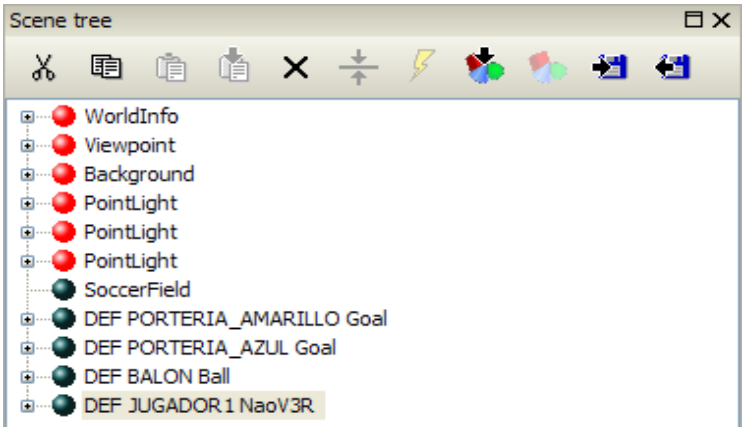


Figura 22. Elementos del árbol de escena

4.3.1. Nodo WorldInfo

Este nodo proporciona el ambiente de ejecución de los movimientos del robot, y está subdividido en los siguientes campos (ver figura 23):

- **Campo información del mundo (*info)*:** en este campo se puede almacenar información referente al mundo (p.ej. para qué se diseñó, quién lo creó y la fecha del trabajo). Al seleccionar este campo, en la parte inferior se encuentra un cuadro de texto, el cual permite cambiar la información al nodo seleccionado.

- **Campo gravedad (*gravity*):** en el campo gravedad existen tres valores X,Y y Z respectivamente. El campo X se refiera a la gravedad que existe a lo largo de la cancha, el campo Y se refiere del suelo hacia arriba y abajo y el campo Z a lo ancho de la cancha. Para mantener la simulación lo más realista posible la gravedad de la tierra se mantiene con un valor de -9.81 en el eje de las Y, mientras que en el eje X y Z se mantiene en cero.

- **Campo de paso de tiempo (*basicTimeStep*):** este campo permite determinar la velocidad en que se ejecutará un movimiento; el valor predeterminado que se usará es de 40 milisegundos.

- **Campo tiempo real (*runRealTime*):** este campo se debe marcar como verdadero para las ejecuciones en tiempo real del robot.

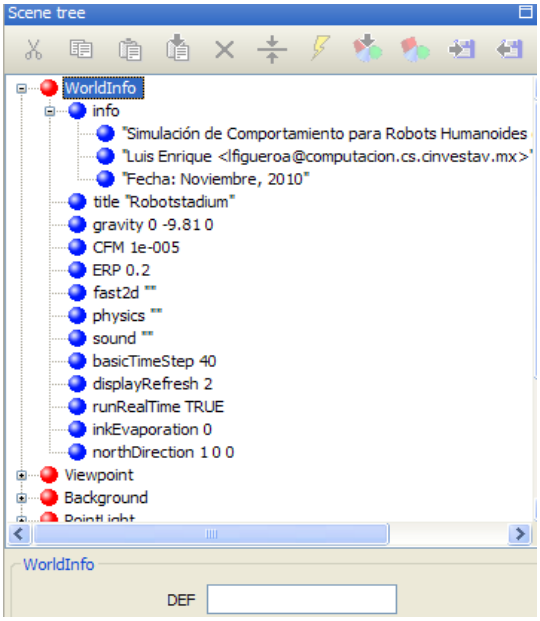


Figura 23. Nodo *WorldInfo* y sus respectivos campos

4.3.2. Nodo Viewpoint.

El nodo *Viewpoint* permite posicionar a la cámara para interactuar con el mundo de la realidad virtual en un punto determinado (ver figura 24). Para ubicar la cámara es necesario conocer los siguientes campos:

- **Campo de visión (*fieldOfView*):** Admite alejar o acercar la cámara del mundo virtual. Si se le otorgan valores negativos, la imagen que proyectaría será una inversa vertical. Los valores adecuados para manejar este campo de visión son de 0.5 a 1.5 para que no sufra de distorsión la cámara.

- **Campo orientación (*orientation*):** este campo permite hacer una rotación a la cámara en los campos X,Y,Z y Alfa, donde X se refiere a lo largo del campo, Y a lo alto del campo, Z a lo ancho del campo y Alfa es un ángulo que tiene la cámara de acuerdo a su posición.

- **Campo posición (*position*):** el campo en cuestión permite hacer una traslación al lugar que se le indique; sus valores están comprendidos en X, Y y Z (tienen las mismas referencias que el campo orientación).

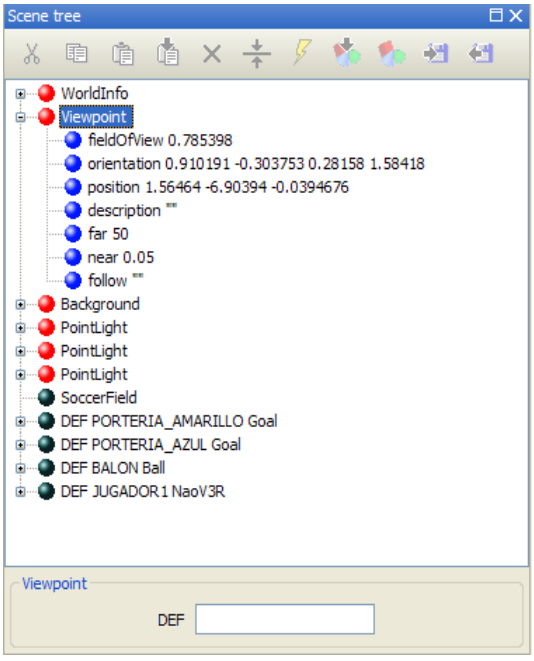


Figura 24. Nodo *Viewpoint* y el contenido de sus campos

4.3.3. Nodo Background.

Este nodo permite establecer un color de fondo gracias al campo que se llama color de cielo (*skycolor*). Por defecto se usara el color cafe, para componer el color cafe utiliza el formato RGB y sus respectivos valores son 0.435294, 0.4, 0.317647 (ver figura 25).

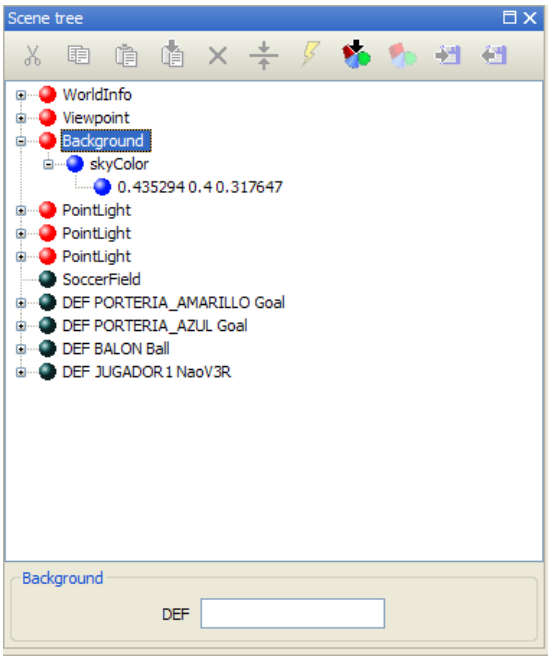


Figura 25. Nodo *Background* para establecer el fondo.

Los tres primeros nodos del árbol de escena (*WorldInfo, Viewpoint y Background*) no pueden ser cortados, copiados o pegados. Una sola instancia de cada uno de estos nodos debe estar presente en cada mundo Webots y en ese preciso orden.

4.3.4. Nodo Pointlight

El nodo *Pointlight* determina la luminosidad, el color, la intensidad, la ubicación de las luces y el radio que abarcará como nodos principales (ver figura 26). Para darle valores al “foco” es necesario conocer los siguientes campos:

- **Campo intensidad del ambiente (*ambientIntensity*):** este campo permite controlar la intensidad del ambiente, obtener una luz más brillante o más opaca dependiendo de lo que se desee.

- **Campo atenuación (*attenuation*):** indica la posición en que la luz se dirige en el campo, por eso tiene tres valores para posicionar la luz en el terreno de juego (X,Y y Z).

- **Campo color:** permite indicar el color de la luz, si la luz es blanca, amarilla, etc.. Por defecto se va a trabajar con la luz blanca cuyos valores son 1,1,1 en formato RGB.

- **Campo intensidad (*intensity*):** permite aumentar la intensidad a la imagen, sin embargo no es recomendable aumentar la intensidad porque los colores se empiezan a perder por la intesidad de la luz.

- **Campo localización (*location*):** ubica la posición del foco en el campo; con los valores X,Y y Z se determina el origen de la luz.

- **Campo encendido (*on*):** este campo da la posibilidad de apagar o encender el foco que se quiere utilizar.

- **Campo de sombra (*castShadows*):** se obtiene sombra de los objetos que están en el campo, por defecto se establece como *False*.

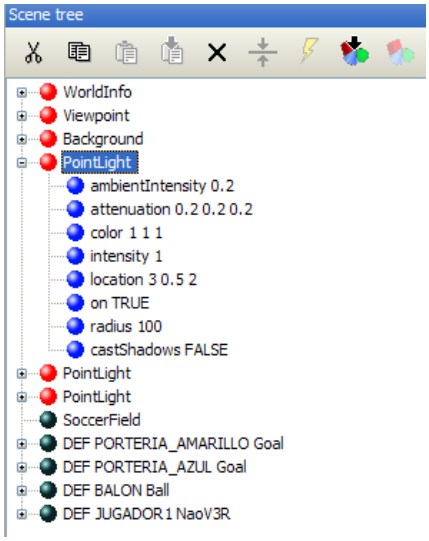


Figura 26. Nodo *Pointlight* para establecer el fondo.

4.3.5. Nodo Goal

Este nodo permite establecer el prototipo de la portería en el terreno del juego con ayuda de VRML. El nodo *goal* tiene cuatro campos (ver figura 27), los cuales se explican a continuación:

- **El campo traslación (*translation*):** este campo permite ubicar a la portería en un lugar del campo. El primer valor corresponde al eje de las X (largo del campo), el segundo valor corresponde al eje de las Y (alto del campo) y el tercer valor corresponde al eje de las Z (ancho del campo). Los valores predeterminados son 3,0,0. Si la porteria se quiere en el lado opuesto de la cancha, sus valores serán -3,0,0.

- **El campo rotación (*rotation*):** este nodo determina la posición que tendrá la portería sobre el lugar establecido, tiene valores X,Y,Z y Alfa. Alfa permite rotar a la porteria sobre su propio eje, mientras que los valores X,Y y Z permiten ubicar una posición en ese eje de la rotación. Por defecto tiene valores de 0 1 0 0. Si se quiere que la porteria este del otro lado, su rotación seria 0,1,0, 3.14159.

- **El campo color de poste (*postColor*):** este campo permite establecer del color que se quiere la porteria. Su color es amarillo para una portería y azúl para la otra.

- **El campo color de soporte (*supportColor*):** la portería tiene un color en los costados de los postes que permite que la portería no se caiga, su color predeterminado es el blanco.

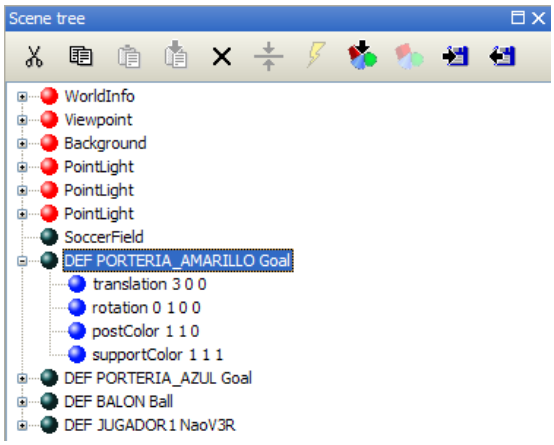


Figura 27. Nodo *Goal* para establecer la ubicación y color de la portería.

4.3.6. Nodo Ball

Este nodo permite localizar la pelota dentro del terreno de juego; no se puede modificar el color de la pelota ni la textura, ya que está definida en el codigo VRML. Para determinar la posición de la pelota se establece mediante su único campo de translación (*translation*) el cual tiene coordenadas X,Y y Z (ver figura 28). La coordenada Y siempre sera 0.042949, para que esté sobre el piso, mientras que las coordenadas X y Z pueden variar.

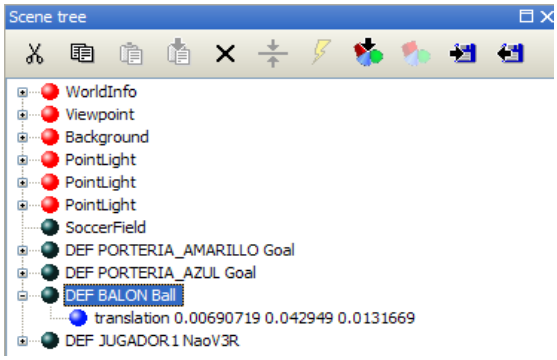


Figura 28. Nodo *Ball* permite determinar la ubicación de la pelota en el campo.

4.3.7. Nodo NaoV3R

Este nodo es el más importante del mundo virtual, ya que determina el prototipo de Nao en VRML. En este nodo se puede ubicar la posición de Nao, el color que este posee, el tamaño de la imagen que se establece en el mundo virtual (ver figura 29) entre otras características que se definen a continuación:

- **El campo traslación (*translation*):** este campo permite ubicar al Nao en un lugar del campo. Los valores que se utilizan son X,Y,Z. El valor por defecto para Y es 0.333629, para que sus pies puedan tener contacto con el piso, mientras que X y Z pueden tomar cualquier valor que esté dentro del terreno de juego (para X de -3 a 3 y para Z de -2 a 2)

- **El campo rotación (*rotation*):** determina la posición que tendrá el robot Nao sobre el lugar establecido en la traslación, tiene valores X,Y,Z y Alfa. Alfa permite rotar al Nao sobre su propio eje, mientras que los valores X,Y y Z permiten ubicar una posición en ese eje de rotación. Por defecto tiene valores de -0.00280348, 0.999992, 0.00278927. Si estos valores son cambiados, la orientación ya no estaría paralelamente al campo, sino que estaría ubicado en otro punto en el espacio.

- **El campo color de Nao(*Color*):** este campo permite establecer el color que se quiere que tenga el Nao. El color por *default* que se utilizará es azúl (su combinación en RGB es 0 0 1).

- **El campo controlador (*controller*):** este campo permite establecer el controlador que utilizará el Nao. Este controlador guarda todos los archivos que se implementarán en el Nao (.cpp, .java, .py, etc). Al seleccionar este campo, en la parte inferior se puede escoger el controlador a utilizar o también editar el código fuente de este controlador presionando el botón Edit. Este automáticamente cargará en la ventana de Editor de Texto el código del controlador seleccionado.

- **El campo controlador de argumentos (*controllerArgs*):** este campo permite llevar un control interno de que robot se esta controlando con dicho proceso mediante el código que se utiliza; se puede observar que este tiene parámetros de -p 54101. A este número se le saca el residuo entre 10, si el valor es diferente de cero, el robot es atacante, si no el robot es defensa.

- **El campo nombre (*name*):** este campo es útil para establecer el control sobre los nombres de los robots que se encuentran en el terreno de juego.

- **El campo posición de ventana (*windowPosition*):** en este campo es posible ubicar la ventana de la cámara dentro del mundo virtual. Para que la cámara se encuentre en la parte superior derecha se establecen valores para X de 1 y valores para Y de 0; si se quiere en la parte superior izquierda, se establecen valores para X de -1 y para Y de 0. Cuando se utiliza más de un robot, hay que ubicar estratégicamente la posición de las ventanas para que no se sobre encimen y el usuario pueda apreciar lo que están viendo los robots en el terreno de juego. Cuando se encuentra de pie determina el tamaño de la imagen que se encuentra en el mundo virtual, por defecto trae el valor de 1 (se recomienda dejar este valor, ya que un tamaño más grande puede perjudicar la visión del usuario en la simulación) y solo acepta números enteros.

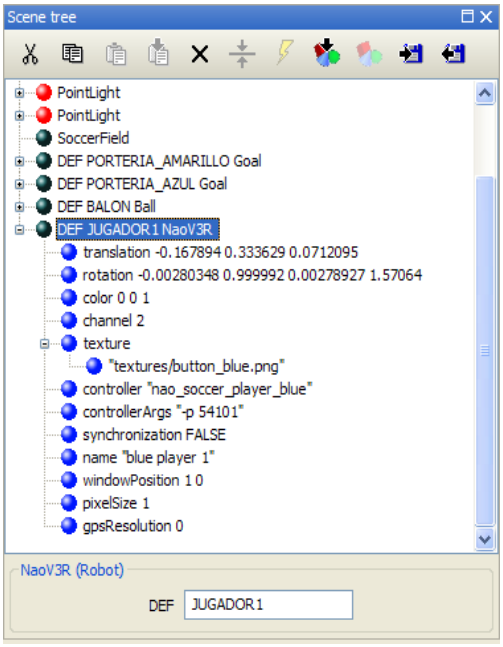


Figura 29. Nodo NaoV3R permite determinar la ubicación, color y otras características del robot Nao.

Capítulo 5

Simulación de juego robótico utilizando Webots

En este capítulo se abordará la programación del robot Nao sobre una plataforma de simulación de Webots. También se describirá la programación de los sensores y locomociones que el robot Nao utiliza en la simulación, así como también la programación de secuencias del juego. Finalmente se describirá el desarrollo de la inteligencia artificial integrada al juego utilizando un autómata de estados finitos no deterministas.

5.1. Programación del Nao sobre una plataforma de desarrollo

La plataforma del desarrollo de la simulación del robot Nao está dividida en cuatro categorías (ver figura 30):

1. **Sistema operativo**. Los diversos sistemas operativos que se utilizan para poder programar la simulación del Nao son Windows, Linux y/o MacOs. Cada uno de estos sistemas operativos pueden representar ventajas para el usuario. En este proyecto de tesis, se utilizó el sistema operativo Windows, porque es un ambiente más cómodo para el usuario, más fácil de instalar (a diferencia de Mac OS y Linux) y tiene la ventaja de poder obtener videos en formato AVI.

2. **Plataforma de desarrollo**. Con el paso de los años han ido apareciendo plataformas de desarrollo que simplifican la programación de aplicaciones robóticas. Estas plataformas ofrecen acceso más sencillo a sensores y actuadores, suelen incluir un modelo de programación que establece una determinada organización del software y permiten manejar la creciente complejidad del código cuando se incrementa la funcionalidad del robot.

   Es posible manipular al robot Nao de forma física o virtual. Para la forma física existen las plataformas de desarrollo de Choreographe y Nao's SDK, mientras que para la forma virtual existen numerosas aplicaciones que se mencionaron en el capítulo 3. En la simulación de esta tesis se escogió la plataforma de desarrollo Webots, principalmente porque es la plataforma de desarrollo oficial para participar en la competencia de simulación de la RoboCup. Además, se eligió porque nos ofrece una robustez en el desarrollo de las aplicaciones, porque cuenta con todos los sensores y actuadores que utiliza el Nao, y también porque es posible lograr una gran precisión en el diseño virtual del robot Nao.

3. **Lenguaje de programación**. Es posible programar el robot Nao en cualquiera de los siguientes lenguajes: C, C++, Python, Matlab. En este proyecto, se escogió el lenguaje C++ ya que es un lenguaje de programación basado en objetos, es mucho más sencillo aprender que otros lenguajes porque la mayoría de los programadores inician con C.

4. **Aplicación**. Las aplicaciones finales que se pueden desarrollar utilizando esta simulación del Nao son varias. Se pueden obtener aplicaciones de visión, reconocimiento de objetos, hacer cálculos de la distancia entre objetos, u obtener el cálculo de trayectoria para el desplazamiento del robot mismo. Es posible hacer reconocimiento de voz utilizando el micrófono, a fin de interactuar con un lenguaje natural del ser humano hacia el robot, para que efectué ciertas acciones. También es posible desarrollar aplicaciones para que el robot ejecute distintos tipos de desplazamiento: en línea recta, en curva, correr, etc. Estas son solo algunas de las muchas aplicaciones que podrían ser programados utilizando el simulador de Webots.

   En nuestro proyecto de tesis, las aplicaciones de la simulación con respecto al desplazamiento del robot utilizan la dinámica y cinemática, para poder percibir la velocidad real del robot así como la velocidad de la pelota.

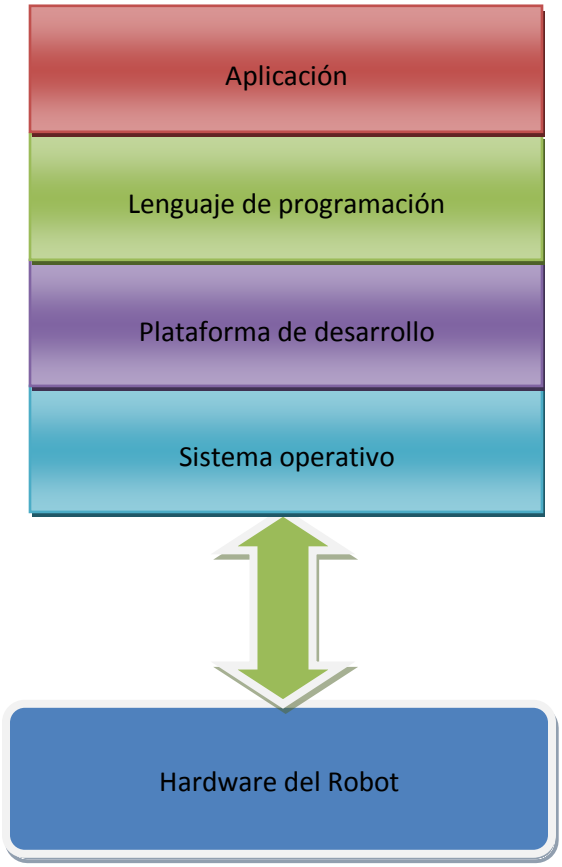


Figura 30. Arquitectura de la plataforma de desarrollo

5.2 Actuadores del Nao en Webots

El mundo virtual que se generó en el capítulo 4 se muestra en la figura 31. Se observa que hay un robot Nao, una pelota y la portería amarilla de frente.

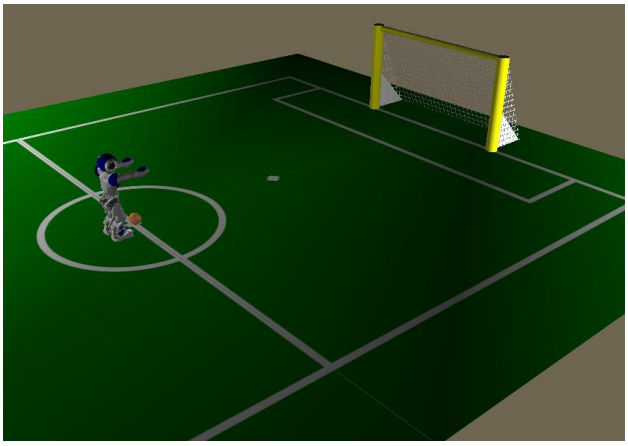


Figura 31. Mundo virtual donde se observa el Nao, la pelota y la portería

Los movimientos que tiene implementados el simulador son:

- Caminado hacia adelante (Forward y Forward50).
- Caminado hacia atrás (Backward).
- Caminado lateral izquierda (SideStepLeft).
- Caminado lateral derecha (SideStepRight).
- Giro a la izquierda de 40 grados (TurnLeft40), de 60 (TurnLeft60).
- Giro a la derecha de 40 grados (TurnRight40), de 60 (TurnRight60).
- Giro de 180 grados (TurnLeft180).
- Tiro (Shoot).

Con ayuda de los servos del robot y del editor de movimiento (figura 32 y 33), se creó la combinación de movimientos necesarios para el desplazamiento del robot. Los nuevos desplazamientos programados fueron los siguientes:

- Incorporación del robot del piso boca abajo (StandUpFromFront)
- Giro del robot en el piso para ubicarlo boca abajo (vuelta)
- Tiro con un alcance de 2.3 metros con pierna derecha (Shoot) y con pierna derecha (Shoot2).
- Tiro lateral izquierda con un alcance superior a 1.5 metros (tiroizq)
- Tiro lateral derecha con un alcance superior a 1.5 metros (tiroder)

- Paso de 2 centímetros (Forwards2)
- Rodear a la pelota hacia la izquierda (lateralIzquierdaI, giroDerecha40, lateralIzquierdaI)
- Rodear a la pelota hacia la derecha (lateralDerechaI, giroIzquierda40, lateralDerechaI)
- Un paso al frente con pierna izquierda (iniciocaminar)
- Paso hacia adelante con pierna derecha y luego con pierna derecha (intermediocaminar)
- Un paso hacia adelante con pierna derecha (final caminar).

Estas locomociones permiten determinar en qué algoritmos se utilizan dichos movimientos. La programación de estas locomociones se encuentra en el Anexo.

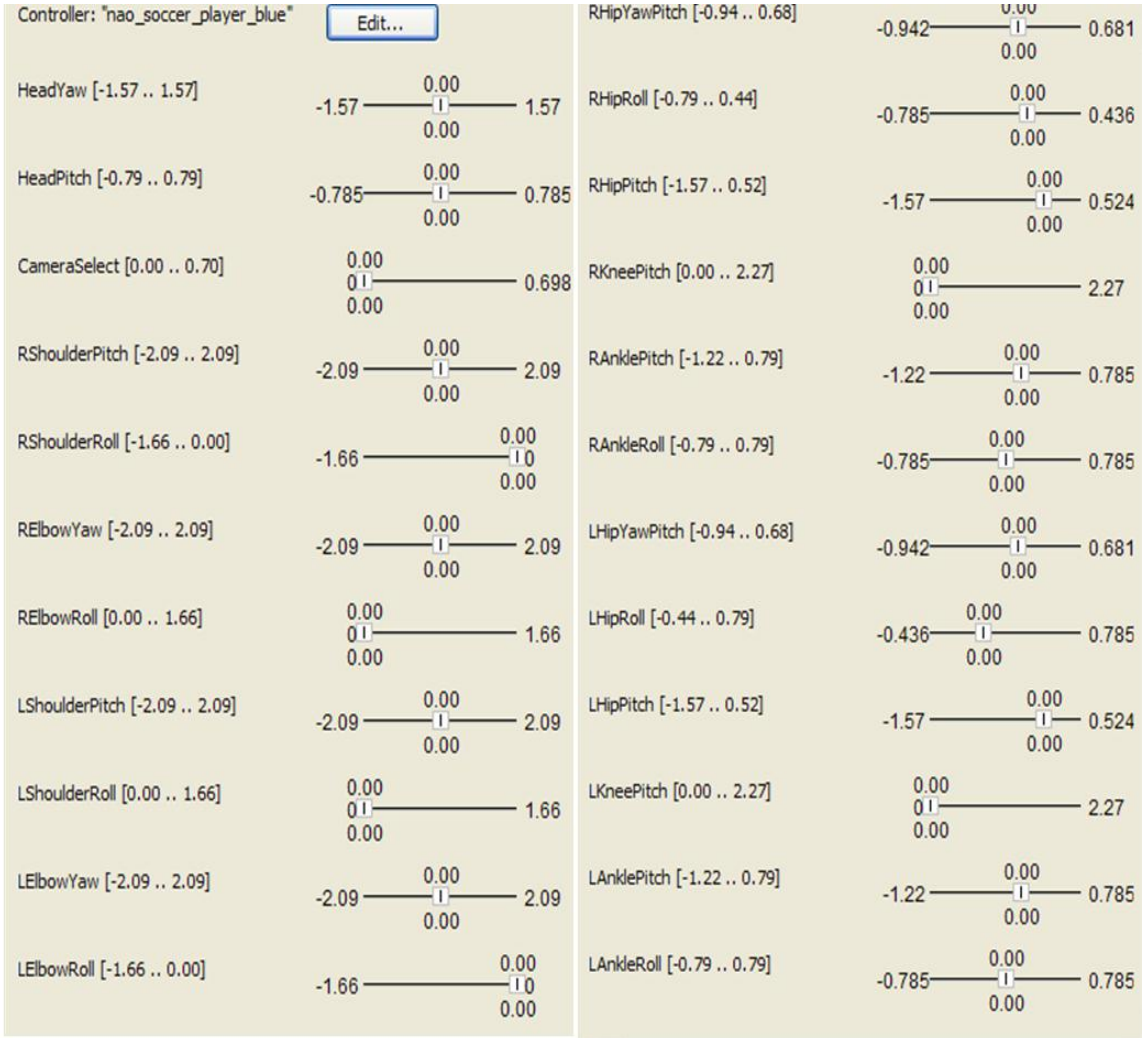


Figura 32. Imagen de la ventana de los servos del robot

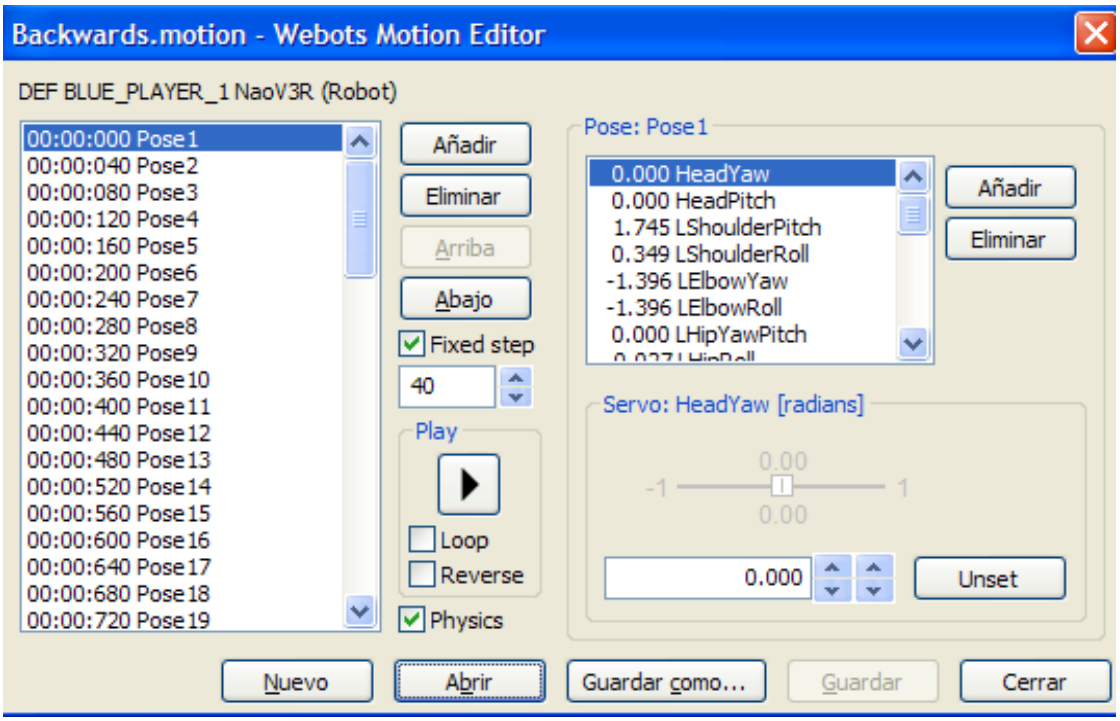


Figura 33. Editor de movimiento con el caminado hacia atrás abierto.

5.3. Sensores del Nao

Como ya se mencionó anteriormente, Nao cuenta con múltiples sensores para desempeñar cada una de sus tareas (ver capitulo 2). Se trabajó principalmente en la utilización de la cámara así como del acelerómetro. La cámara se utiliza principalmente para la visión y el acelerómetro para determinar la posición del robot.

5.3.1. Acelerómetro.

Un acelerómetro es un dispositivo para medir aceleración y fuerzas inducidas por la gravedad a la que un cuerpo está sometido. Lo hace mediante la medición de fuerzas y se utiliza la segunda ley de Newton F=ma (donde F es fuerza, m es masa y a es aceleración), despejando la fórmula para obtener la aceleración se obtiene la siguiente formula a= F/m.

Un objeto estático está sometido a la fuerza de la gravedad, la cual va a detectar el acelerómetro. La aceleración de la gravedad es 9.81 m/s^2 y en la dirección vertical (z) se procederá a leer un valor diferente debido a la inclinación.

El acelerómetro con que cuenta el Nao tiene tres ejes (X, Y, Z). Con la información anterior, el eje que nos importa conocer por su inclinación es Z.

El acelerómetro permite calcular si el robot está de pie (Z es superior a 5), si está en el suelo boca abajo (Z está en un rango de 0.7 a 5) o boca arriba (con los valores de Z menores a .7). Todos estos valores son mostrados en la tabla 2.

X	Y	Z	Resultado
-0.2 a 0.2	-0.01 a 0.01	**> 5**	El robot está de pie
>9	< 0	**< = 5**	El robot está caído
>9	< 0	**.7 a 5**	El robot está caído boca abajo
>9	< 0	**Menor de .7**	El robot está caído boca arriba

Tabla 2. Resultados para determinar la posición del robot

5.3.2. Cámara.

La cámara que posee el Nao permite capturar imágenes simuladas de su entorno. La resolución es de 160 pixeles de ancho por 120 pixeles de alto. Tiene un campo de visión (*FOV* o *Field Of View*) de 46.4 grados (0.81 radianes). El campo de visión es el ángulo lineal o de área del mundo observable que es visto en un momento dado (Figura 34).

Para obtener el FOV vertical, se aplica la siguiente fórmula:

FOV Vertical = FOV* alto/ancho.

Con los valores anteriores, se sustituye en la formula:

FOV Vertical = 0.81 * 120/160

FOV Vertical = 0.6075 radianes o 34.8°

El FOV vertical tiene un valor final de 34.8° o 0.6075 radianes con los valores que tiene la cámara del Nao en la simulación.

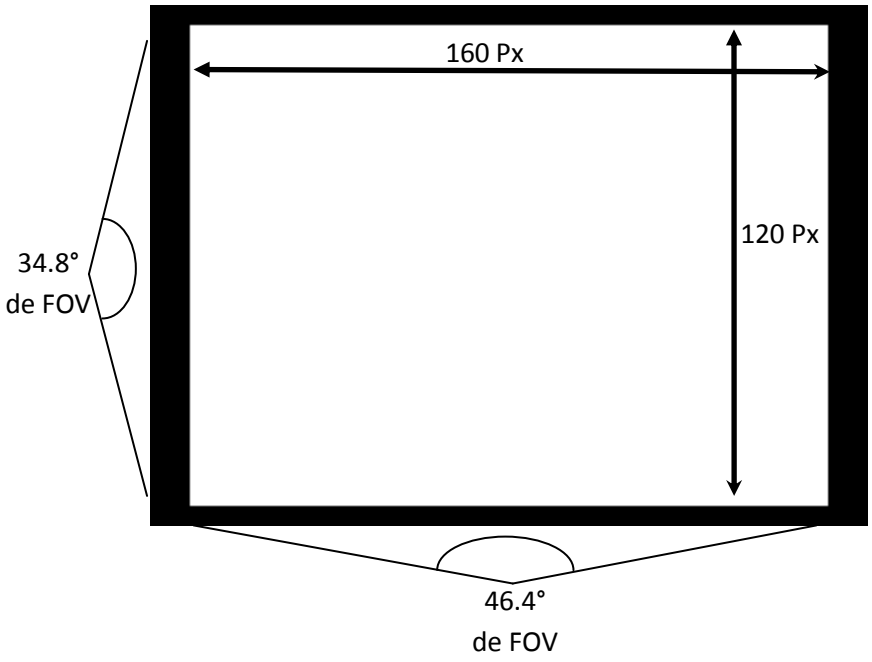


Figura 34. Resolución y ángulo de visión del Nao

5.4. Programación del simulador para el juego robótico

Las tareas que fueron programadas sobre el robot Nao para el juego robótico fueron las siguientes:

- Programación de los actuadores del robot Nao
- Programación de los sensores del Nao
  - Acelerómetro
  - Cámara
- Programación de secuencias para el juego:
  - Levantar el robot
  - Ubicar y centrar la pelota
  - Caminar hacia la pelota
  - Ubicar la portería
  - Tirar la pelota hacia la portería

5.4.1. Programación de secuencias para el juego.

El objetivo del robot Nao es jugar futbol y meter gol en la portería contraria. Para lograr este objetivo, el robot debe llevar a cabo una serie de pasos. Primeramente debe encontrarse en posición vertical y ubicar su posición dentro del campo de juego. Después, el robot debe encontrar la pelota y dirigirse hacia ella. Una vez que tiene enfrente a la pelota, debe ubicar la portería contraria, para caminar con la pelota hacia allá. Una vez que el robot llega cerca de la portería contraria, debe disparar hacia la portería. Si en la ejecución de alguno de los pasos anteriores, el robot llega a caerse, su prioridad principal es levantarse y redirigirse al primer paso.

5.4.2. Levantarse

El robot Nao, en cualquier momento de su caminado puede llegarse a caer. Por esta razón, se debe implementar una estrategia para poder levantarlo. En este caso, se utilizó un algoritmo de Webots para levantarlo del piso utilizando el código de acelerómetro.

5.4.2.1. Algoritmo para levantar al robot.

Mediante el acelerómetro es posible determinar si el robot está de pie o caído. Si el robot está caído, se debe determinar si se encuentra boca arriba o boca abajo. Si este se ubica boca arriba, tiene que rodar para poder ubicarse boca abajo. Por el contrario, si el robot se localiza boca abajo, tiene que efectuar la locomoción de levantar para poder estar de pie. El algoritmo que efectúa esta tarea esta descrito la figura 35.

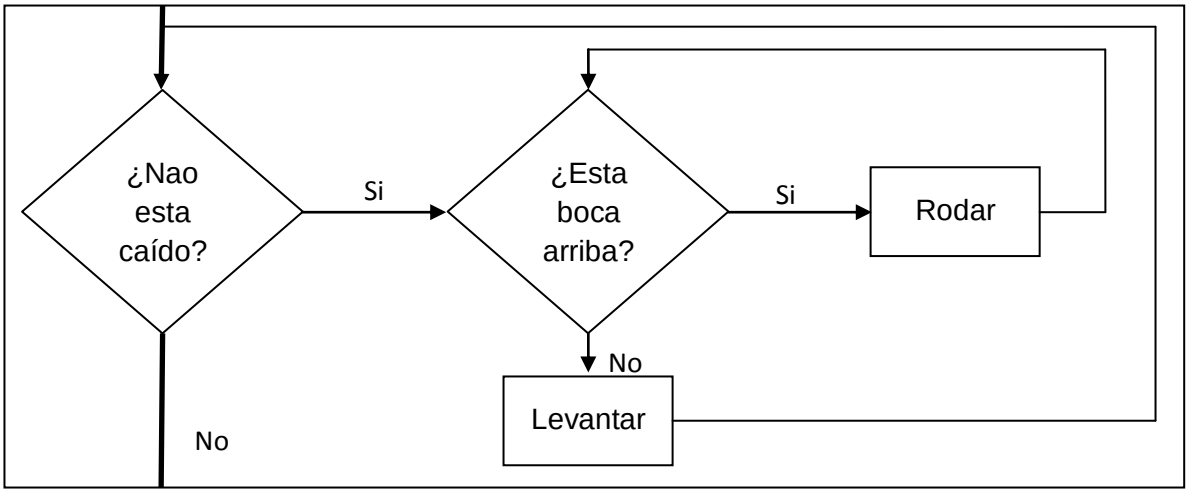


Figura 35. Diagrama de flujo que determina si el robot está de pie o caído.

5.4.2.2. Implementación del acelerómetro y locomoción de rodar y levantar en Nao

El simulador de Webots para el robot Nao trae las librerías necesarias para utilizar el sensor de acelerómetro.

La librería que se necesita para la utilización del acelerómetro es:

#include <webots/Accelerometer.hpp>

Ya incluida la librería dentro del código de simulación, es necesario la inicialización del acelerómetro, para lo cual se necesita crear un objeto para la utilización de la clase acelerómetro:

accelerometer = getAccelerometer("accelerometer");

La palabra *accelerometer* es el nombre de la variable, *getAccelerometer* es para mandar a llamar la función de la clase, y *accelerometer* es el nombre del sensor a utilizar.

Para habilitar el acelerómetro se utiliza la siguiente sintaxis:

accelerometer->**enable**(SIMULATION_STEP);

Donde *accelerometer* es el nombre de la variable creada anteriormente, *enable* permite habilitar el acelerómetro y SIMULATION_STEP es una variable en milisegundos que permite ser actualizada en ese tiempo.

La obtención de los valores del acelerómetro se realiza mediante la siguiente instrucción:

acc = accelerometer->**getValues**();

Donde *acc* almacenará un arreglo de tres valores (X, Y y Z), el valor que interesa es el tercero del arreglo.

El código que se utilizó para el acelerómetro está en el apéndice de esta tesis.

Si el valor de *acc* en el valor de Z es inferior a 5, significa que el robot Nao está tirado, pero se tiene que verificar si está caído boca arriba o boca abajo. Si el valor de Z es menor o igual de .7, significa que el robot está caído boca arriba, por lo que se tiene que hacer que el robot ruede primero con la locomoción de rodar (vuelta) para que el robot esté boca abajo. Si es mayor de .7 significa que el robot

Nao está boca abajo, por lo que se llama a la locomoción levantar (*StandUpFromFront*).

Para habilitar estos movimientos se necesita crear una nueva locomoción con la siguiente sintaxis:

vuelta = new **Motion**("../motions/vuelta.motion");

levantarSuelo = new **Motion**("../motions/StandUpFromFront.motion");

Donde *vuelta* y *levantarSuelo* es el nombre de las variables, "*new motion*" es la palabra reservada para crear un nuevo movimiento y finalmente "../motions/*.motion" es la ruta donde se encuentra guardado el movimiento.

Para ejecutar los movimientos de levantar del suelo se utilizan las siguientes instrucciones en Webots.

playMotion(vuelta);

playMotion(levantarSuelo);

5.4.3. Ubicar y centrar la pelota

Para la ubicación de la pelota, es necesario conocer los valores del color naranja en su escala RGB (ver tabla 3). Con el color de la pelota identificada, se necesita ejecutar un algoritmo para identificar a la pelota en la imagen tomada. Se utiliza un segundo algoritmo para obtener diferentes imágenes del campo. Estas distintas imágenes permitirán distinguir cuando se requiere ubicar la pelota, la portería, o los caminos que debe seguir el robot. Un tercer algoritmo se encarga de verificar que la pelota esté siempre delante del robot y en el centro de la imagen, una vez que esta haya sido detectada.

Pelota/Color	Red	Green	Blue
Naranja	1	0.5	0.2

Tabla 3. Valores RGB de la pelota naranja

5.4.3.1. Algoritmos para ubicar la pelota.

El algoritmo descrito en la figura 35 permite ubicar la pelota dentro de la cancha de juego. Este algoritmo permite determinar si la pelota se encuentra o no en la imagen (ver la figura 36), mediante la captura de una imagen de la cámara. Después entra en un ciclo en el cual se recorre pixel por pixel buscando que el pixel que se está analizando sea de color naranja. En caso de ser naranja el pixel, se guardan sus coordenadas X y Y de ese valor y se procesa el siguiente pixel. En caso de que el pixel no sea de color naranja, se obtiene el siguiente pixel y se sigue buscando pixeles hasta terminar con todos los que hay en la imagen. Cuando se termina de procesar toda la imagen, el algoritmo determina cuantos pixeles en la imagen son de color naranja. En caso de no encontrarse pixeles naranjas se analiza otra imagen. Si se encuentran pixeles naranjas en la imagen, se determina el valor medio de X y el valor medio de Y, y estos valores se almacenan en memoria para poder procesarlos posteriormente en el algoritmo de dirección y de distancia de la pelota.

El algoritmo de la figura 37 hace un barrido de la imagen para localizar la pelota. En este algoritmo, si la pelota es encontrada en la primera imagen, se procede a salir del algoritmo. En caso contrario, primeramente efectuará un recorrido de derecha a izquierda con la cámara superior del Nao, tomando una foto y procesándola. En caso de que la pelota siga sin ser encontrada, se pasa a recorrer de izquierda a derecha con la cámara inferior tomando una imagen y procesándola. En este caso, si la pelota sigue sin ser localizada, se procede a realizar la locomoción de media vuelta (turnLeft180) y empezar de nuevo con el ciclo hasta que la pelota se encuentre.

El algoritmo de la figura 38 detalla el algoritmo para detectar la pelota. Una vez ubicada la pelota, se procede a poner la pelota a la mitad de la cámara.

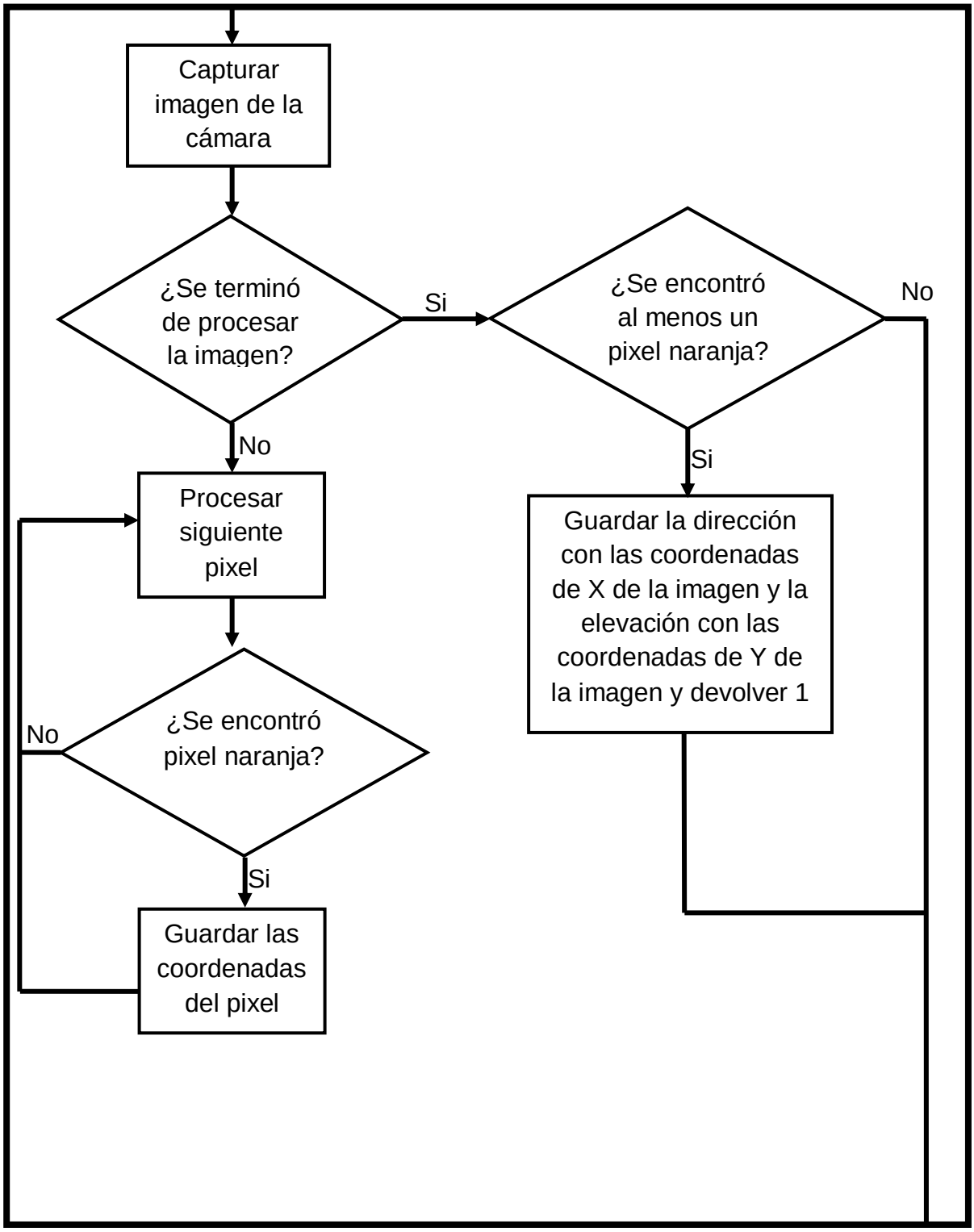


Figura 36. Diagrama de flujo para determinar si la pelota se encuentra en la imagen

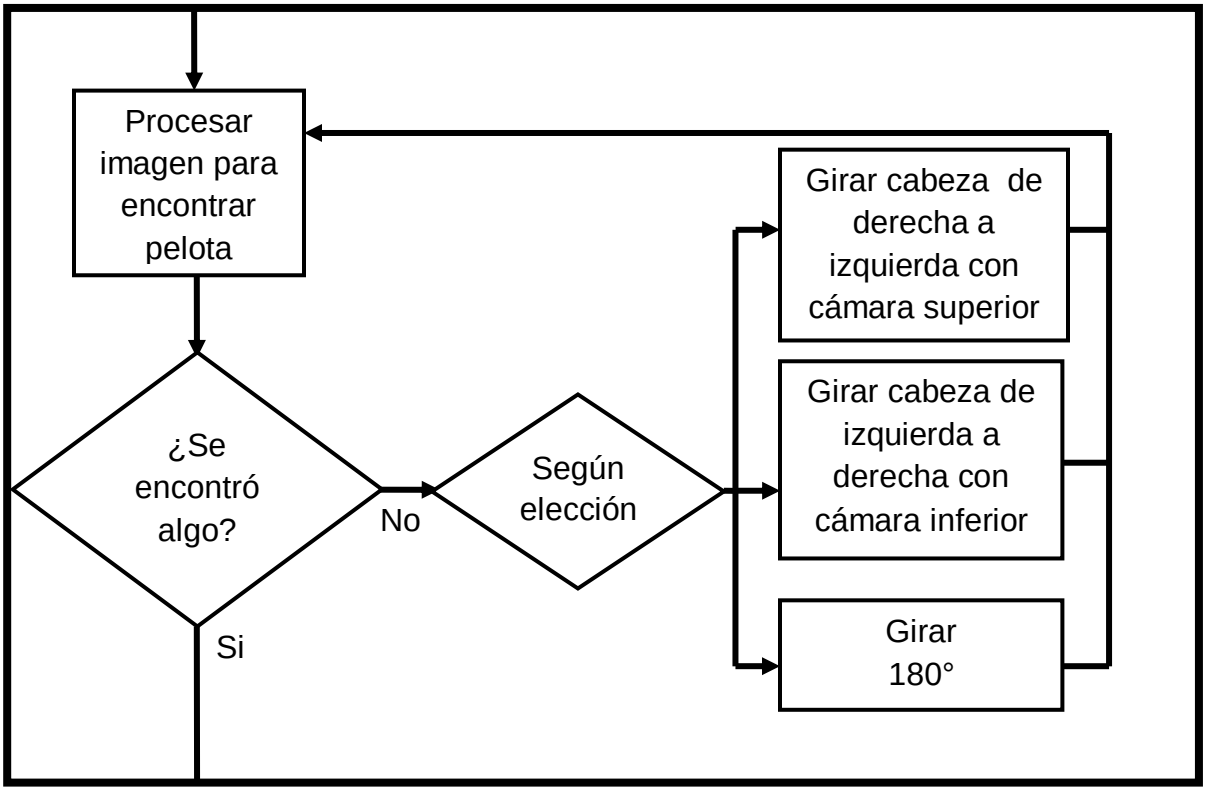


Figura 37. Diagrama de flujo para detectar la pelota

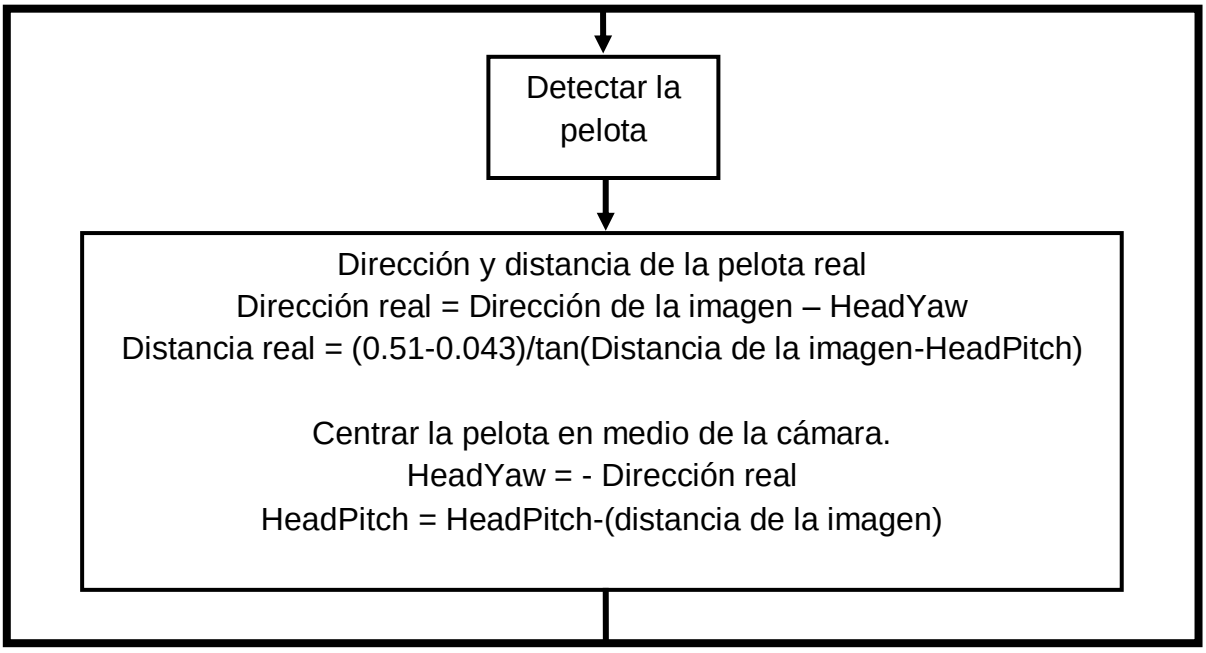


Figura 38. Diagrama de flujo para centrar la pelota

5.4.4. Caminado hacia la pelota

La secuencia del caminado hacia la pelota está clasificada en dos partes: Cálculo de la distancia y dirección de la pelota y locomoción hacia la pelota.

1. Cálculo de la distancia y dirección de la pelota

En esta parte es necesario determinar que la dirección que puede tomar la pelota en una imagen va de -0.40 radianes a 0.40 radianes (aproximadamente de -23° a 23°) y es el valor de X que se guardó previamente. Si la pelota se encuentra casi en el centro de la pantalla tiene un valor que oscila entre -0.1 a 0.1; si la pelota se encuentra en el lado derecho de la imagen tiene un valor superior de 0.1 y si la pelota se encuentra en el lado izquierdo de la imagen tiene un valor inferior de -0.1 (ver figura 39).

La distancia que puede tomar la pelota va de -0.39 radianes a 0.39 radianes. Si la distancia es negativa, significa que la pelota se encuentra en la parte de abajo de la imagen, pero si la distancia es positiva, la pelota se ubica en la parte superior de la imagen. La distancia es el valor de Y que se guardó en el algoritmo de buscar a la pelota. Sin embargo, la dirección y distancia que se obtienen solamente son respecto a la imagen; falta calcular la dirección real y la distancia real en el terreno de juego en donde se encuentra la pelota.

La dirección real se obtiene de la diferencia de la dirección de la imagen menos la posición del HeadYaw (HeadYaw es el servo que gira de izquierda a derecha en el cuello). Así se obtiene el valor real de la pelota, el cual tiene valores que van desde 1.91 radianes hasta -1.91 radianes (110° a -110° aproximadamente).

La distancia real del robot a la pelota se obtiene de la diferencia de la altura real del robot (0.51 metros) y el diámetro de la pelota (0.043 metros) sobre la tangente de la diferencia de la distancia de la imagen menos *HeadPitch* (*HeadPitch* es el servo del cuello que permite moverlo de arriba hacia abajo). La distancia real está expresada en metros.

El caminado hacia la pelota parece una tarea relativamente sencilla, pero en realidad es más complejo de lo que uno se imagina, porque mientras esté caminando, tiene que estar observando la pelota; si la pelota en un momento dado se mueve p.ej. cuando un robot haya golpeado la pelota y cambiado su dirección, tiene que recalcular su caminado y no llegar hacia la dirección que se le había asignado antes.

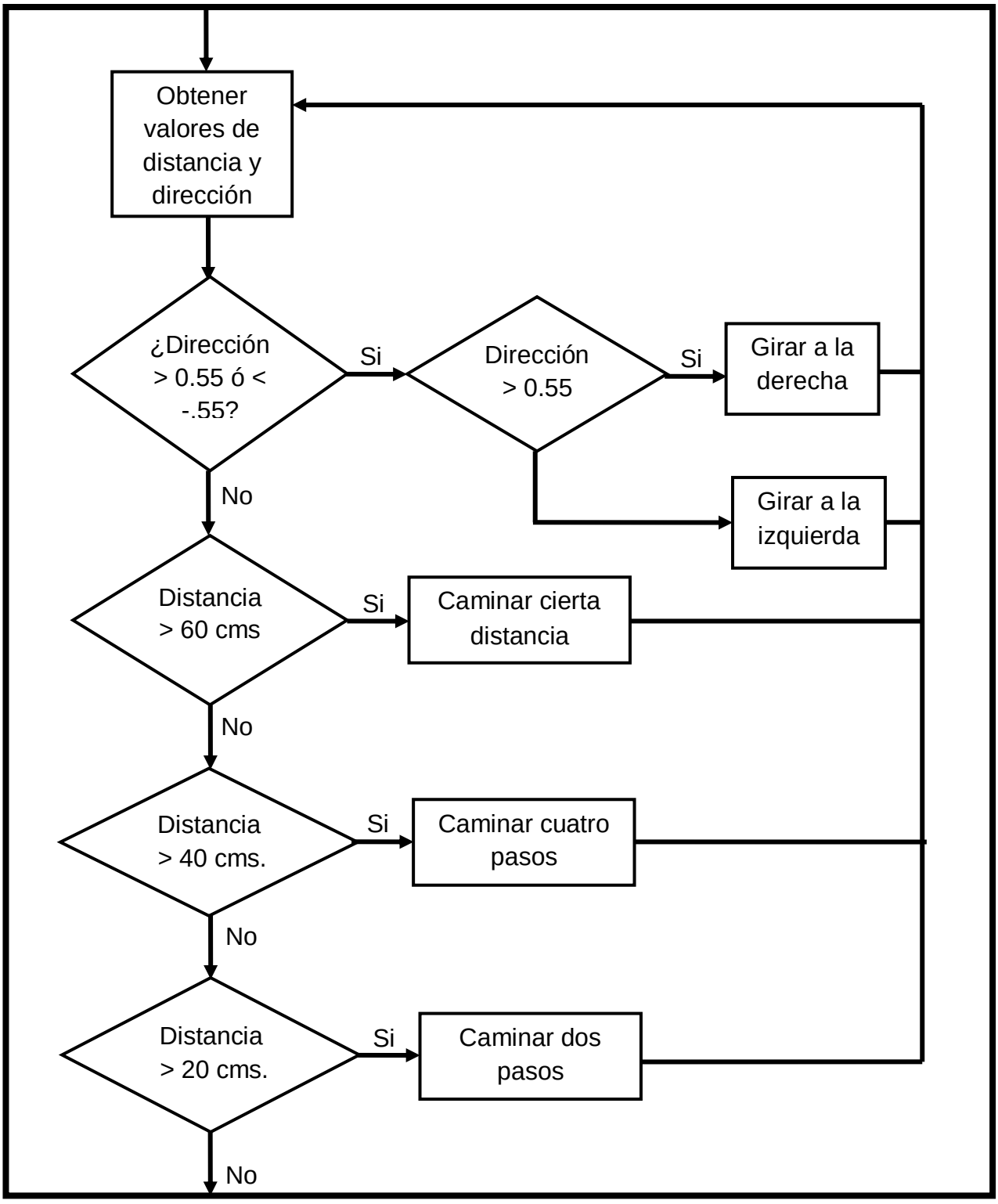


Figura 39. Diagrama de flujo para caminar hacia la pelota

2. Locomoción hacia la pelota

Para la segunda parte del caminado hacia la pelota, dependiendo de los valores de la dirección y distancia obtenidos anteriormente, se utilizan las siguientes locomociones:

- **Girar a la derecha.** Si la dirección de la pelota es superior a 0.55 radianes, se realiza la locomoción de giro a la derecha de 40 grados (turnRight40)
- **Girar a la izquierda.** Si la dirección de la pelota es inferior a -0.55 radianes, se efectúa la locomoción de girar a la izquierda 40 grados (turnLeft40).
- **Caminar dos pasos.** Si la pelota se encuentra a una distancia mayor de 20 cms. y menor de 40 cms. se procede a realizar la locomoción de dos pasos (Forward) que aproximadamente son 17 cms.
- **Caminar cuatro pasos.** Si la pelota se encuentra en una distancia mayor de 40 cms y menor de 60 cms, se procede a realizar la locomoción normal (Forward50).
- **Caminar más de 60 centímetros.** Si la pelota se encuentra a una distancia mayor de los 60 cms., se procede a realizar la combinación de tres locomociones (iniciocaminar, intermediocaminar y finalcaminar):
    - La primera locomoción un paso hacia al frente permite al robot dar un paso hacia adelante con la pierna izquierda.
    - La segunda locomoción permite al robot dar un paso hacia al frente con la pierna derecha y posteriormente con la pierna izquierda. Esta locomoción se repetirá “n” veces hasta que haya alcanzado la distancia deseada.
    - La tercera locomoción permite acomodar la pierna derecha a la par de la pierna izquierda y así finalmente estar con las dos piernas juntas.

5.4.5. Ubicar la portería

Esta actividad del robot Nao consiste en ubicar a la portería del equipo contrario. Para ubicar la portería, se utiliza la cámara superior del robot. La Librería NaoCam de Webots permite procesar la imagen para identificar el objeto que estamos buscando. El objeto a ubicar (con su correspondiente posición) es la portería amarilla.

Para utilizar la cámara, es necesaria la utilización de la siguiente librería:

#include "NaoCam.hpp"

El constructor para esta clase se llama NaoCam. Este constructor se inicializa por defecto al mandar a crear el objeto. Los valores que se mandan son el nombre del robot, la portería contraria y la declaración para poder crear el robot.

new NaoCam(name, NaoCam::SKY_BLUE, (Robot)this);*

Para llamar a la cámara y habilitarla se utiliza el siguiente código:

camera = (NaoCam)getCamera("camera");*

camera->enable(CAMERA_STEP);

La primera línea crea el objeto con el nombre de *camera* y la segunda línea habilita la cámara para su uso.

Para utilizar la cámara superior de Nao, se utiliza:

camera->selectTop();

Para manipular la cámara inferior de Nao se utiliza la siguiente instrucción en Webots:

camera->selectBottom();

Estas líneas permiten intercambiar entre la cámara superior del Nao con la cámara inferior y viceversa.

Para la ubicación de la portería, es necesario saber si el robot meterá gol en la portería azul o en la portería amarilla. Los valores RGB (*Red, Green, Blue* o Rojo, Verde, Azul) del color de la portería se muestran en la Tabla 4.

Portería/Color	Red	Green	Blue
Azul	0.1	1.0	1.0
Amarilla	1.0	1.0	0.1

Tabla 4. Colores RGB de la portería azul y amarilla

Para capturar una imagen con la cámara del Nao, se utiliza la siguiente instrucción:

image = getImage();

Donde *image* es la variable de tipo *char* que guardara la imagen que ha sido tomada con la cámara.

Una vez obtenida esta imagen, será procesada. Cada carácter de la imagen, contiene tres valores, que son los correspondientes a RGB. Por lo consiguiente se va comparando cada pixel con sus respectivos elementos de color RGB para determinar si corresponden al color que se está buscando. En total existen 19,200 pixeles (160 pixeles x120 pixeles) en una imagen y 57,600 que componen en total los colores RGB. Para ubicar la portería, solamente se necesita procesar en el mejor de los casos 160 pixeles (como se explicó en el capítulo 4, solamente hay que procesar la línea media de la imagen para encontrar ambos postes) para ambos postes o 278 pixeles para un solo poste.

5.4.5.1. Algoritmos para ubicar la portería

El algoritmo para determinar si la portería se encuentra en la imagen se describe en la Figura 40. Este algoritmo procesa la imagen y permite reconocer si contiene un poste, dos postes o ningún poste. Es posible determinar si el poste reconocido es el izquierdo o el derecho. Sin embargo, este algoritmo es una pequeña pieza dentro del algoritmo para encontrar la portería (ver figura 41).

Este algoritmo permite determinar si se encontró una portería y determinar si se encontraron ambos postes. En caso de que se hayan encontrado ambos postes se procede a guardar las posiciones de los postes. En el caso de no haber encontrado ningún poste, se realiza un movimiento de la cabeza de izquierda a derecha y se procesa de nuevo otra imagen. Si después de esto no encuentra ni un poste, efectúa su movimiento de derecha a izquierda. Si aun sigue sin encontrarse ni un poste, ejecuta un giro de 180° para volver a empezar nuevamente en la búsqueda de la portería. Por otro lado, si el robot encuentra un poste, si este es el izquierdo, procede a buscar el poste restante girando su cabeza hacia la derecha. Si lo encuentra guarda los valores, si no encuentra el otro poste, se determina que el poste izquierdo está en su lado derecho, por lo que se le indica que gire 180° para poder ubicar ambos postes correctamente. Si es el poste derecho el que encuentra, procede a hacer una posición similar tomando en cuanta que el próximo poste a buscar es el izquierdo.

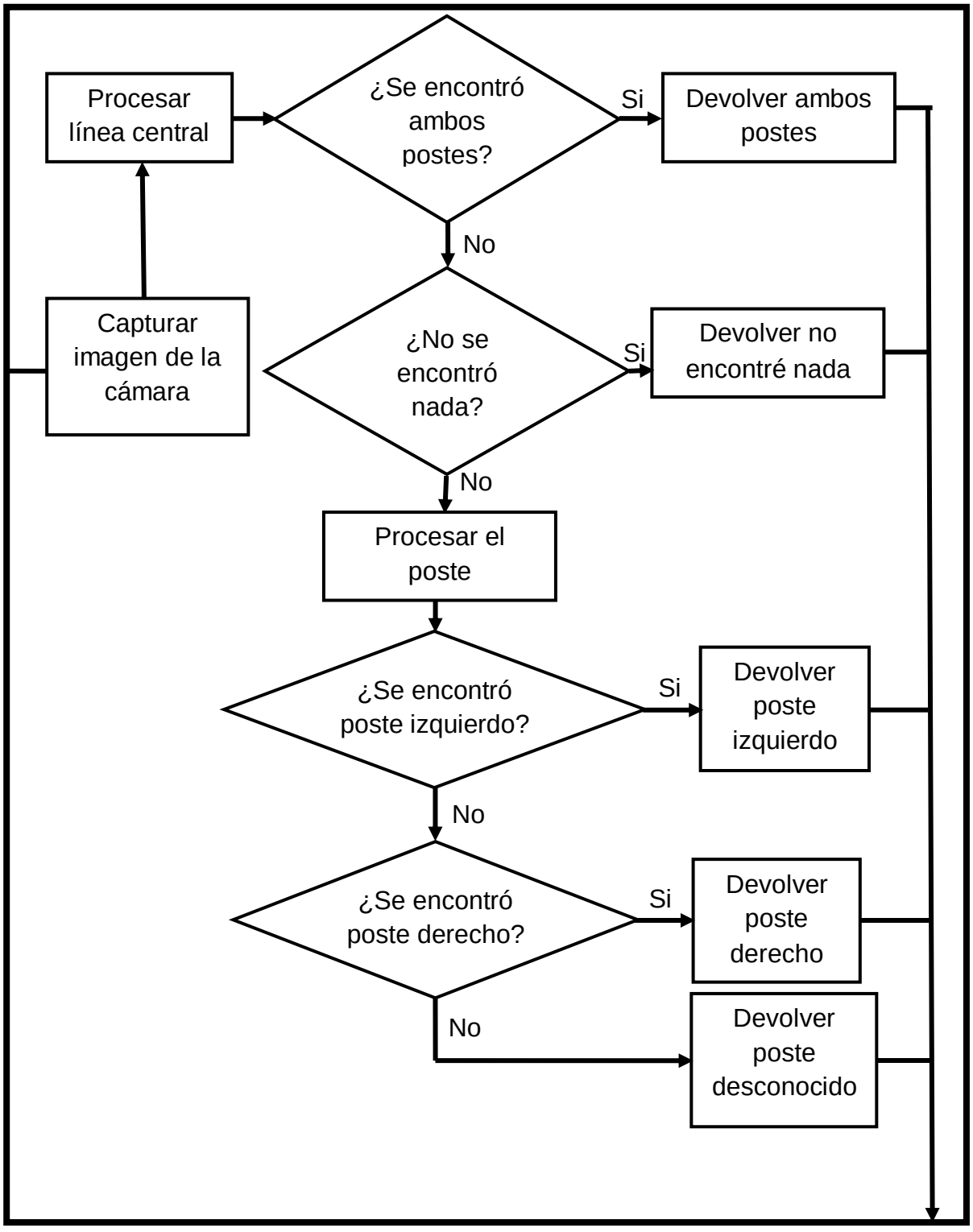


Figura 40. Diagrama de flujo para determinar si la portería se encuentra en la imagen.

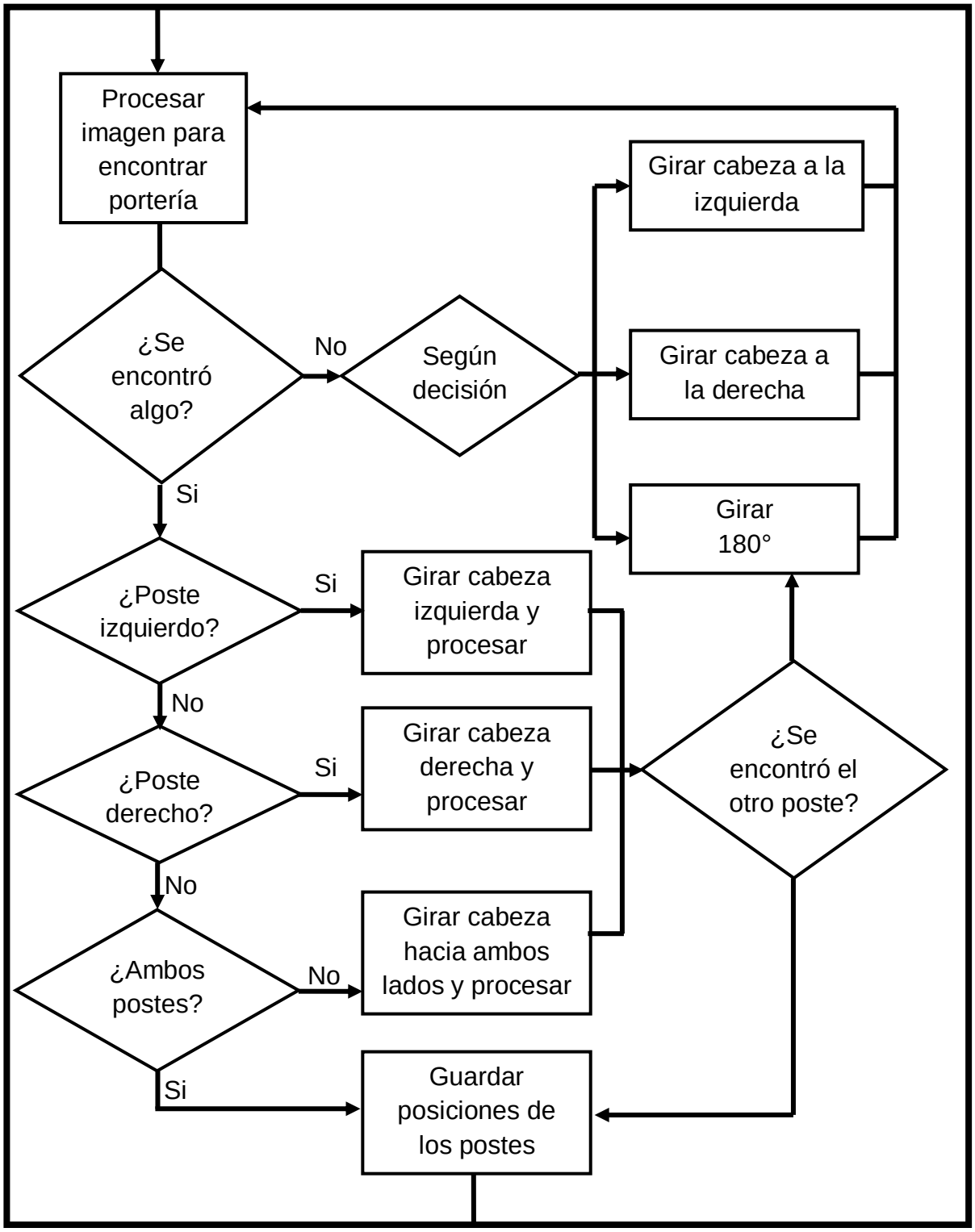


Figura 41. Diagrama de flujo para localizar ambos postes.

5.4.6. Rodear a la pelota

La operación de rodear a la pelota es efectuado por el robot cuando requiere posicionarse frente al balón para patearlo y poderlo meter en la portería. Cuando el robot se encuentra enfrente a la portería con la pelota enfrente (a menos de 20 centímetros), tiene que tomar la decisión de si está en el lugar correcto para poder efectuar el disparo, o “acomodarse” para poder realizar su tiro. El algoritmo para acomodar el robot (ver figura 43) se explica a continuación.

Para calcular si está acomodado o no, el robot levanta la cabeza y ubica la portería de acuerdo a su última posición. En caso de no encontrar la portería, la vuelve a tratar de ubicar nuevamente de izquierda a derecha. Si no la detecta, entiende que la portería podría estar atrás de él mismo y tendría que rodear a la pelota 180° para poder estar de frente y nuevamente hacer el cálculo de la portería.

Al detectar la portería, tiene que proporcionar cierto ángulo donde se encuentre la portería. Para obtener este ángulo en radianes se hace una suma del poste izquierdo más el poste derecho y se divide entre dos. Si la portería tiene un ángulo final entre -0.30 a 0.30, significa que la portería está delante del robot, en el ejemplo de la figura 36.a se aprecia que el ángulo del poste izquierdo es de -0.12 y el ángulo del poste derecho es de 0.4, la suma de estos ángulos da 0.28, dividiendo este resultado entre 2, nos da el ángulo final de 0.14, lo cual significa que la portería está enfrente del robot y puede efectuar su tiro hacia adelante. Si la portería tiene un ángulo superior de 1.20, significa que la portería está en el lado derecho del robot, por lo tanto debe realizar un tiro lateral con la pierna izquierda de izquierda a derecha; la portería da un valor inferior de -1.20, lo que significa que la portería está en el lado izquierdo del robot y debe efectuar un disparo lateral con la pierna derecha de derecha a izquierda (ver figura 42).

Ahora, si la portería está en los ángulos faltantes (0.30 a 1.20 y -0.30 a -1.20) no puede efectuar un tiro lateral ni un tiro de frente, ya que la portería se encuentra en un punto ciego. Para corregir este problema, se debe rodear a la pelota un ángulo aproximado de 45 grados hacia la izquierda o derecha dependiendo del ángulo que se presente.

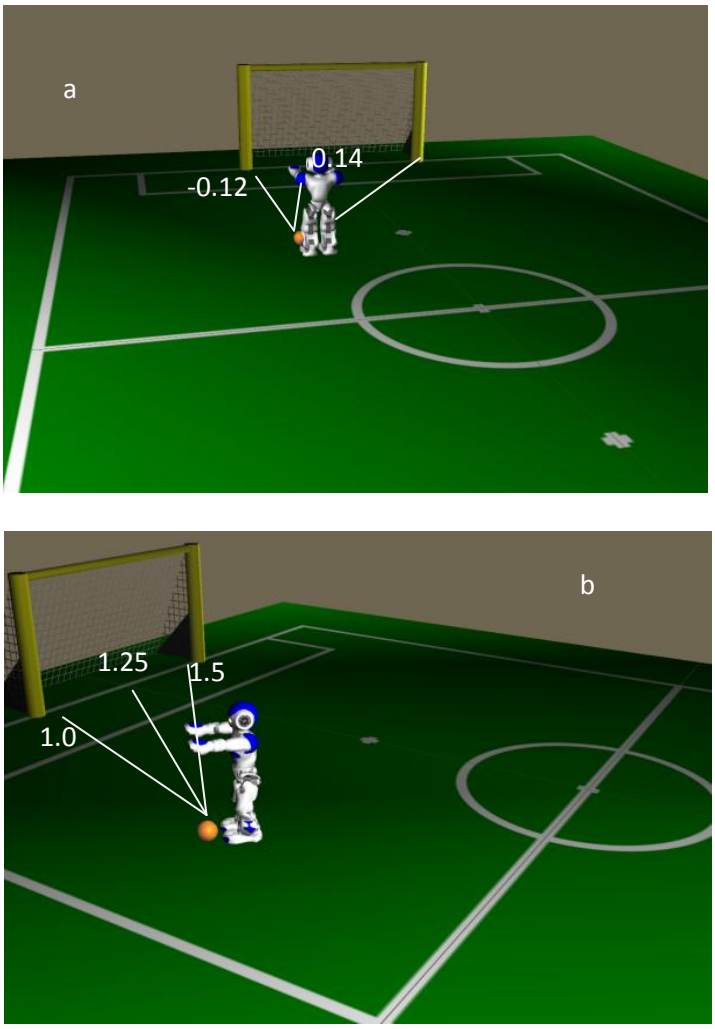


Figura 42. Obtención del ángulo final de la portería: a) cuando la portería se encuentra casi enfrente del robot. b) cuando la portería se encuentra en la lateral

Si la portería tiene un ángulo intermedio de 0.35 a 0.80, se le indica al robot rodear a la pelota a la izquierda, para poder ubicarse de frente y disparar de frente. Si la portería tiene un ángulo intermedio de 0.80 a 1.20, se orienta hacia el lado derecho, para poder ubicarse lateralmente y hacer un disparo lateral con la pierna izquierda; así mismo, se invierten las acciones con signo negativo, por ejemplo si el ángulo está en un rango de -0.35 a -0.80, se rodea a la pelota por la derecha para ubicarse de frente y disparar hacia adelante; si tiene un ángulo de -0.80 a -1.20 se ataca a la pelota por el lado izquierda y se dispara lateralmente con la pierna derecha. Cada vez que se determina que clase de tiro se va a efectuar, este se guarda en una variable para facilitar el algoritmo de tiro que se explicará posteriormente.

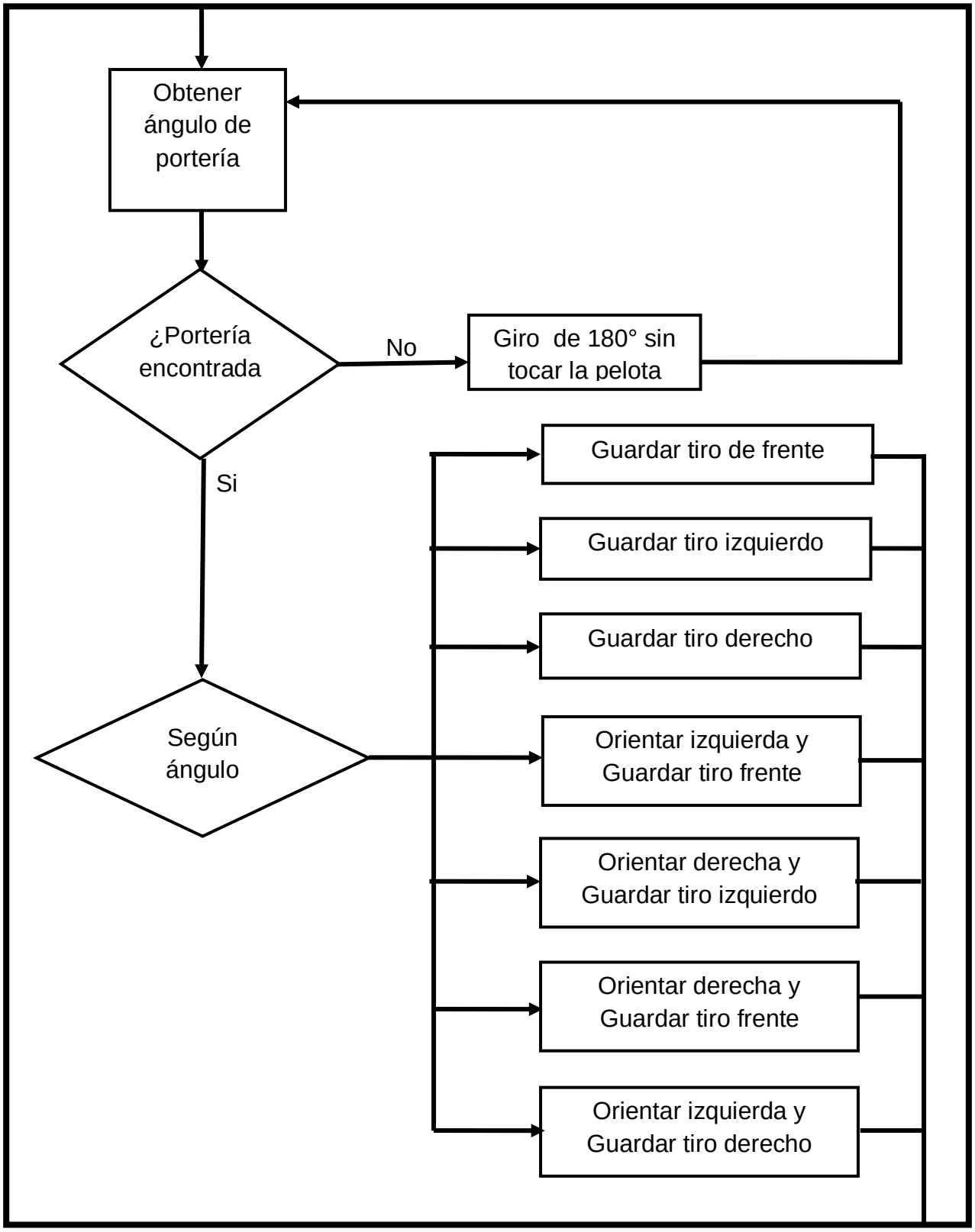


Figura 43. Diagrama de flujo para acomodar el robot en posición de tiro

5.4.6.1. Locomoción para rodear a la pelota

Cuando el robot se encuentra frente a la portería, este requiere rodear a la pelota para poder patearla. Para poder rodear a la pelota, se necesitan las siguientes locomociones para conseguirlo:

- **Rodear a la pelota hacia la izquierda o a la derecha.** Para rodear a la pelota hacia la izquierda, se utilizan dos locomociones que ya están en el simulador. Primeramente se realiza una locomoción de caminado lateral izquierdo (*SideStepLeft*); el robot Nao efectúa dos pasos laterales hacia su izquierda, posteriormente se realiza la locomoción de giro de 40 grados hacia la derecha (turnRight40); en este momento la pelota se encuentra un poco lejos y hacia la izquierda del robot; lo que procede finalmente es efectuar una última locomoción de caminado lateral a la izquierda para posicionar al robot enfrente de la pelota. Para rodear la pelota del lado derecho, solo se invierten las locomociones (si es caminado lateral izquierda es a la derecha y así sucesivamente).

- **Paso de dos centímetros.** La pelota está a más de 7 centímetros del robot, por lo que se procede a caminar a ella con pasos relativamente cortos hasta que la distancia sea menor de 7 centímetros y poder realizar algún tiro de los que se mencionaron previamente.

5.4.7. Tiro

Esta es la última secuencia de juego y finalmente el robot está en posición de tiro, sabe exactamente con que pierna tiene que efectuar el tiro gracias al valor que se almacenó en la secuencia de acomodar para disparo y la dirección que tomará la pelota con respecto al tiro que realice. El algoritmo de la figura 40 explica que primeramente tenemos que acercarnos a la pelota a menos de 15 centímetros. Dependiendo del tiro que se va a realizar, se acomoda la pelota en el respectivo pie para disparar. Si es tiro de frente, tiene que ubicar la pelota enfrente de uno de los pies (escoge el pie más cercano) y si es un tiro lateral, posiciona la pelota a la mitad de ambos pies para poder efectuar ese tiro (ver figura 44).

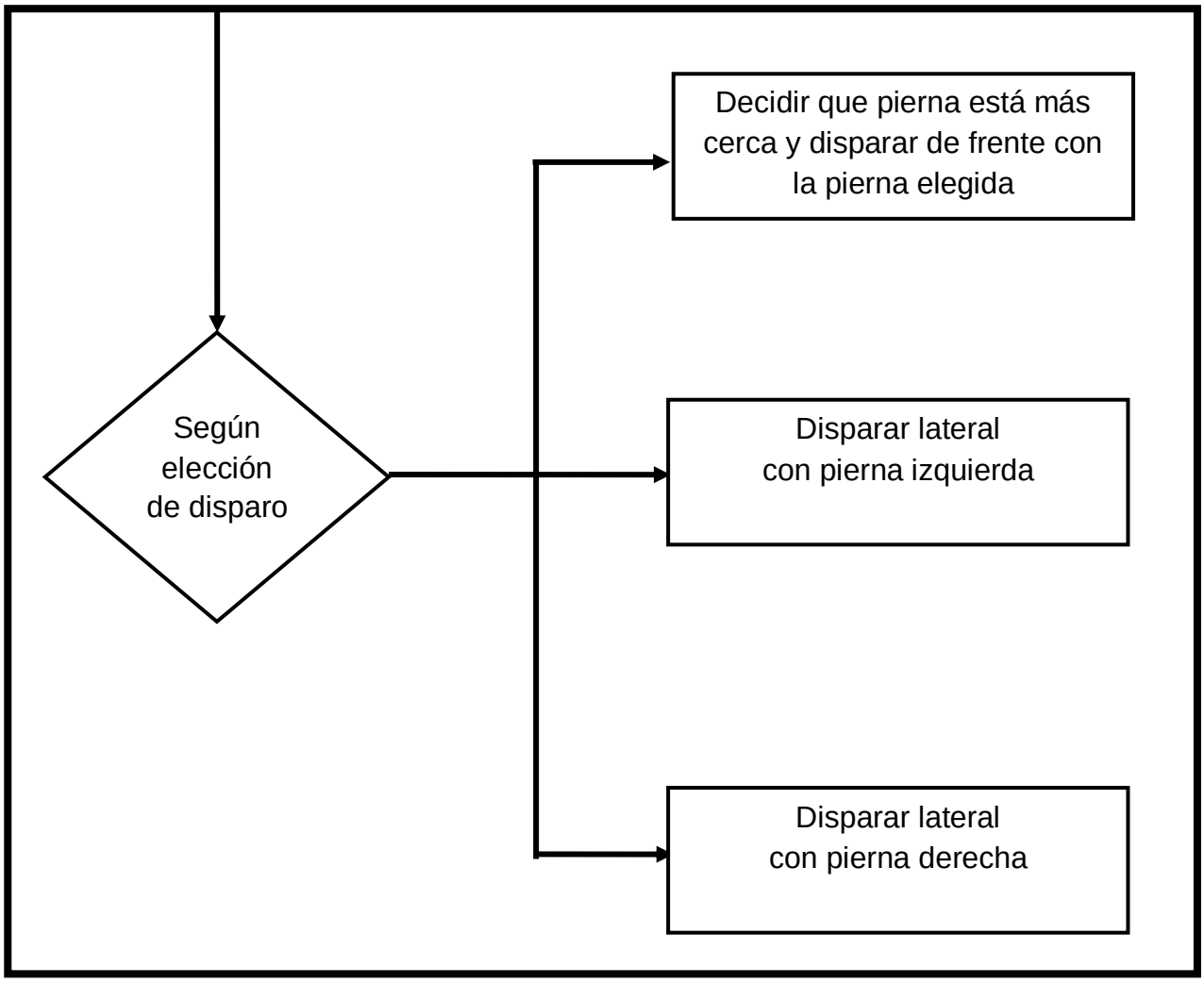


Figura 44. Diagrama de flujo para acercar y elegir el tiro adecuado

5.4.7.1. Locomoción para el tiro

Para realizar un tiro el robot tiene a su disposición las siguientes locomociones:

- **Tiro de frente con pierna derecha o izquierda.** La pelota se encuentra en el lado izquierdo del robot, el robot efectuará un tiro de frente con la pierna izquierda. Esta locomoción permite patear a la pelota hacia adelante con una distancia máxima de 2.3 metros. Cuando se encuentra en el lado derecho, patea con la pierna derecha.

- **Tiro lateral izquierda.** La pelota se encuentra enfrente y cerca del robot y la portería en el lado derecho. Realiza un tiro de costado con la pierna izquierda. Tiene un alcance superior a 1.5 metros.
- **Tiro lateral derecha.** La pelota está enfrente y a menos de 7 centímetros del robot y la portería se encuentra en el lado izquierdo, entonces se procede a realizar un tiro de costado con la pierna derecha. Este tiro tiene una distancia mayor a 1.5 metros.

5.5. Autómata del Nao

El robot Nao está compuesto por dispositivos sensores y actuadores y su correspondiente software para el control de cada sensor y actuador. Con base en la información que proporcionan estos dispositivos el simulador del robot contiene un software que le permite efectuar distintas acciones (caminar, patear, etc). Este software de control se basa en técnicas de inteligencia artificial para decidir lo que el robot tiene que hacer dependiendo del ambiente que perciba.

En esta tesis diseñamos un software de control para el simulador del robot que consiste en un autómata de estados finitos no determinista (AFND) (ver figura 45). Este autómata cuenta con seis estados y diez transiciones para poder logra su objetivo (ver tabla 5).

Estados	Transiciones
1. Encontrar portería	a. Portería no encontrada
2. Encontrar pelota	b. Portería encontrada
3. Caminar a la pelota	c. Pelota no encontrada
4. Acomodar posición	d. Pelota encontrada
5. Disparar	e. Perdí pelota
6. Levantar	f. Pelota cerca
	g. Robot posicionado
	h. Pateo exitoso
	i. Caído
	j. Levantado

Tabla 5. Nombre de los estados y transiciones del AFND

La sintaxis del autómata es la siguiente:

El conjunto finito de estados: <S = {1,2,3,4,5,6}

El alfabeto finito de etiquetas: $\sum = \{a,b,c,d,e,f,g,h,i,j\}$

Conjunto finito de aristas: A= {(1,a)=1, (1,b)=2,(1,i)=6,(2,c)=2, (2,d)=3,(2,i)=6,(3,f)=4,(3,e)=2,(3,i)=6, (4,e)=2, (4,g)=5,(5,i)=6, (5,h)=1,(6,j)=1}

Estado inicial: s*k*=1 >

Cada estado y cada transición se explican a continuación.

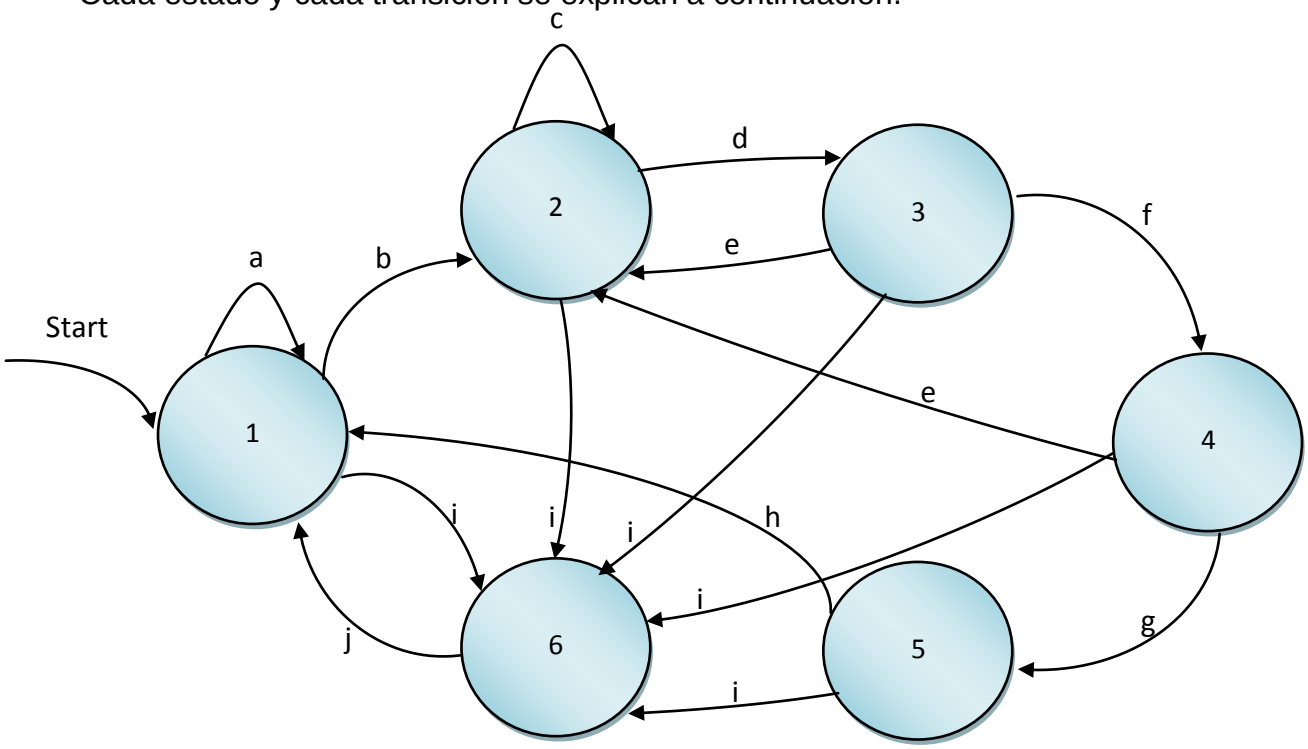


Figura 45. Autómata del Nao Aldebarán

Estado 1: **Encontrar portería**. En este estado el robot se encarga de buscar la portería del equipo contrario. En el juego, las porterías tienen diferente color (azul o amarillo) y el robot sabe al principio del juego cual portería es la del equipo contrario y la de su equipo. Se optimizó el algoritmo de búsqueda de portería que tiene por default el robot.

La visión del campo de un robot para localizar la portería se muestra en la Figura 46. Note que, si colocamos al robot en cualquier parte del campo teniendo su referencial en *alpha* en 1.5708 radianes (*alpha* está definida en el simulador

como 2π), y girando la cabeza podemos obtener capturas de uno, ningún o ambos postes.

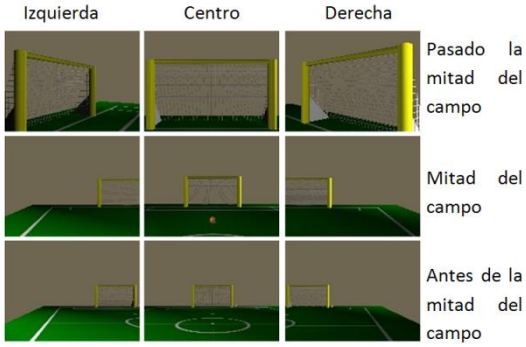


Figura 46. Visón del campo del robot usando la cámara 1

Adicionalmente, al obtener estas imágenes, es posible darse cuenta de que no es necesario procesar toda la imagen, sino que sólo se necesita procesar la mitad de esta para poder ubicar ambos postes [5] (ver figura 47). Sin embargo, aun el procesamiento de la mitad de la imagen lleva también una considerable cantidad de tiempo, por lo cual se mejoró ese algoritmo mediante un barrido en el punto intermedio de la altura y la cual es recorrida en todo lo ancho (Figura 48).

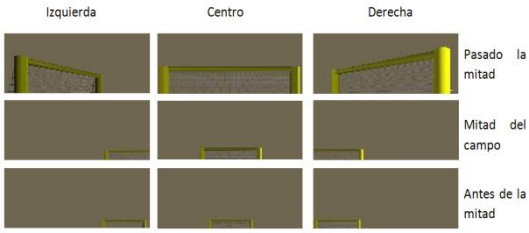


Figura 47. Visión del robot solamente de la mitad de la imagen

En la Figura 48 se describen las posibles imágenes que el robot Nao puede visualizar. En la figura 48.a. se aprecia que el robot encontró un poste y localizó el travesaño en el lado derecho, de lo cual se deduce que encontró el poste izquierdo. En la figura 48.b. el robot encuentra ambos postes por lo que ya no es necesario hacer un barrido vertical. En la figura 48.c. el robot encuentra el travesaño en el lado izquierdo, por lo tanto se deduce que el poste está en el lado derecho.

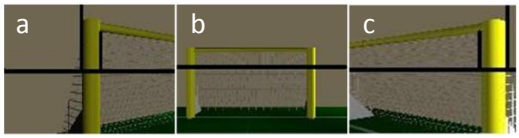


Figura 48.a. Se encontró poste izquierdo. 48.b. Se encontraron ambos postes. 48.c. Se encontró poste derecho

Transición a. **Portería no encontrada**. Esta transición se encarga de pasar del estado 1 al mismo estado. El estado 1 no fue capaz de encontrar la portería del enemigo, por lo tanto en esta transición se le ordena al robot efectuar movimientos (giro de 40, 60 o 180°) para poder cumplir el objetivo del estado 1.

Transición b. Portería encontrada. Esta transición permite pasar del estado 1 al estado 2 si se encontró la portería. El estado 1 encontró la portería del enemigo, así qué guarda la dirección en donde se encuentra la portería enemiga para utilizarla posteriormente.

Estado 2: Encontrar pelota. En este estado el robot se encarga de localizar la pelota. El robot sabe en qué dirección se encuentra la portería contraria, por lo que ahora se le ordena buscar la pelota. En este caso, lo primero que tiene que hacer es girar su cabeza de derecha a izquierda utilizando la cámara 1, extrayendo 5 imágenes porque el robot tiene un ángulo de visión de 0.81 radianes que equivalen a un poco más de 46°. En la Figura 49 se pueden observar las 5 tomas que realiza el robot. De un ángulo total de visión del robot de 46°, este puede visualizar 23° a su izquierda y 23° a su derecha. Con esto se permite obtener un control total de la cancha con solo 5 imágenes. Con cada imagen se utiliza un algoritmo de segmentación para encontrar los píxeles de color naranja. Cabe mencionar que como la intensidad de la luz está controlada, es más sencillo encontrar la pelota en este ambiente simulado.

Se hace un barrido de la imagen de arriba a abajo buscando concentraciones de color naranja, y en caso de que se encuentren, el siguiente paso es localizar el centroide de la pelota. Si la pelota es encontrada, se procede a encontrar la dirección y la distancia.

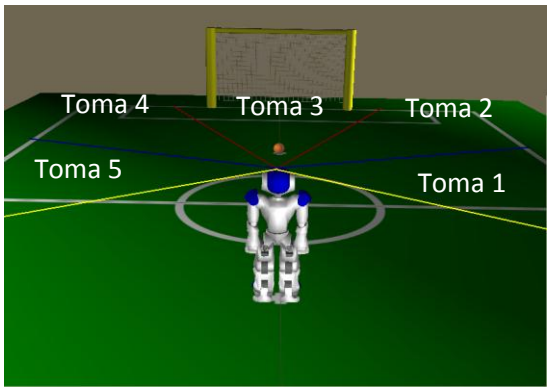


Figura 49. Angulo de visión del Nao

La dirección del robot con respecto a la pelota se calcula obteniendo el ángulo en el que se encuentra le pelota con respecto al plano de la imagen menos la orientación del cuello (HeadYaw) del Nao. En la Figura 50 se observa que el plano de la imagen es de -0.0613125 radianes, mientras que la orientación del cuello es de -1.54327 radianes, por lo cual la dirección exacta de la pelota es de 1.48196 radianes (84° aproximadamente) con respecto al robot.

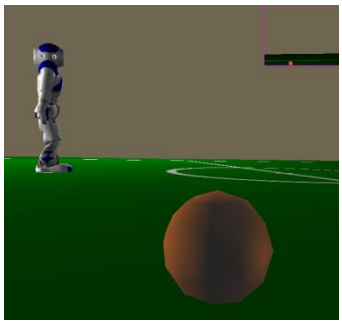


Figura 50. Dirección de la pelota

La distancia del robot a la pelota [6] se obtiene con un cálculo trigonométrico, conociendo el ángulo **α** presente entre el eje óptico y el plano frontal del robot y la distancia **h** que corresponde a la altura de este eje óptico con respecto al suelo que es 0.51 metros (ver Figura 51).

Transición c. Pelota no encontrada. Esta transición permite pasar del estado 2 al mismo estado, si no se encontró la pelota. En este caso, el robot da dos pasos hacia atrás y vuelve al estado 2 y si sigue sin encontrarla, gira 180 grados y vuelve de nuevo al estado 2. Se repiten de nuevo los primeros pasos hasta que encuentre la pelota.

Transición d. Pelota encontrada. En esta transición se pasa del estado 2 al estado 3 y se ejecuta cuando la pelota es encontrada. Si el robot finalmente pudo localizar la pelota, guardará la dirección y la distancia de la pelota.

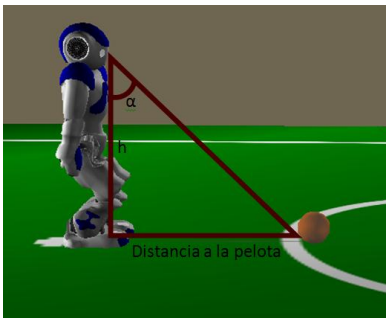


Figura 51. Calculo de la distancia del robot a la pelota.

Estado 3: Caminar a la pelota. En este estado el robot se tiene que desplazar hacia la pelota. Con la pelota ubicada, el siguiente estado a seguir es caminar hacia la pelota. Para ello necesitamos la distancia y la dirección de la pelota que se obtuvo en la acción buscar pelota.

Si la dirección está en el rango de -0.2 radianes a 0.2 radianes (11° aproximadamente) significa que la pelota está al frente del robot. Si la dirección es menor de -0.2 la pelota se encuentra a nuestra izquierda y si es mayor de 0.2 la pelota se encuentra a la derecha.

Se desarrolló un algoritmo para poder caminar una distancia corta para llegar a la pelota rápidamente. Si la pelota se encuentra a 1.5 metros, se le ordena al robot que camine aproximadamente 1.4 m. Dependiendo del ángulo al que está la pelota, el robot gira hacia el lado izquierdo o el derecho.

Transición e. Pelota perdida. Esta transición pasa del estado 3 al estado 2 si se extravía la pelota. Se pasa a ésta transición cuando el robot en un momento determinado pierde la pelota de su vista y tiene que volver al estado 2 para volverla a localizar.

Transición f. Pelota cerca. En esta transición se pasa del estado 3 al 4 cuando la pelota está cerca del robot. Para poder pasar a esta transición, el robot tiene que estar a menos de 20 centímetros y menos de 20 grados de la pelota.

Estado 4: Acomodar posición para disparar. En este estado se acomoda el robot enfrente de la pelota para poder efectuar su disparo. Este estado es el más delicado de todos. Se empieza acomodando al robot de forma tal que la pelota se encuentre enfrente de él con un ángulo menor a 15 grados, de tal forma que la pelota se encuentre enfrente de los pies del robot. Luego se visualiza la portería para tratar de buscarla en la posición en donde la habíamos localizado previamente. Si no se encuentra la portería, el robot rodeará a la pelota en un ángulo de 180° y buscará de nuevo la portería. Si la portería se encuentra en el angular de -0.35 a 0.35 radianes significa que la portería está enfrente. Si la

portería supera los 1.20 radianes la portería está del lado derecho; si la portería es inferior de -1.20 radianes se encuentra del lado izquierdo. Si la pelota no se encuentra en esos rangos, se aproximará a la dirección que se encuentre más cerca, por ejemplo, si está a 0.80 radianes el ángulo se considera como 1.20 en lugar de 0.35.

Para poder posicionar el robot se creó una nueva locomoción para orientarlo hacia la izquierda o hacia la derecha. Lo que se consigue con esta locomoción es que el robot pueda rodear a la pelota sin que la toque. Para poder rodearla por su lado derecho, el robot ejecuta dos pasos laterales a su derecha, un giro a su izquierda de 40 grados y luego dos pasos laterales también a su derecha. En caso contrario, se efectúan dos pasos laterales a su izquierda, un giro de 40 grados a su derecha y luego dos pasos laterales también a su izquierda.

El robot puede quedar en cualquiera de las tres posiciones para tiro (ver figura 52); y dependiendo de la posición en la que se encuentre efectuará un tiro distinto.

Transición g. Robot posicionado. Esta transición pasa del estado 4 al estado 5 cuando el robot está enfrente de la pelota. Finalmente tenemos al robot delante de la portería y con ángulo de tiro y se almacena el disparo a realizar en el estado 5.

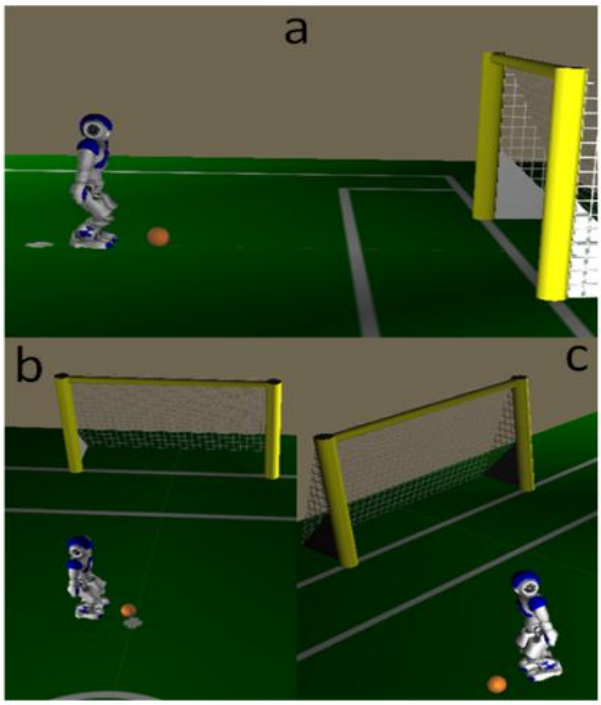


Figura 52.a. De frente a la portería. Figura 52.b. Portería en el lado izquierdo. Figura 52.c. Portería en el lado derecho.

Estado 5: Disparar. En este estado se patea la pelota. El robot se encuentra ya en el último estado “ordinario” (porque el estado 6 solamente ocurrirá si se cae el robot). En este estado el robot tiene que decidir cuál de los tres tiros realizar: tiro normal, tiro lateral izquierdo o tiro lateral derecho. Dependiendo del valor guardado en el estado 4, será el tiro a realizar.

En nuestra implementación, el tiro normal incrementó su trayectoria de 1.3 metros a 2.2 metros. Los tiros laterales pueden superar los dos metros, con respecto a la implementación por *default* de Webots.

Transición h. Pateo exitoso. En esta transición se pasa del estado 5 al estado 1. Aquí el robot finalmente pudo completar un ciclo de su estado y pateó la pelota con éxito hacia la portería contraria.

Transición i. Caído. En esta transición el robot pasa de cualquier estado al estado 6. Hay varias situaciones bajo las cuales el robot puede caerse. El robot pudo haber caído mientras realizaba alguna de las siguientes actividades:

- buscando a la portería,
- buscando la pelota,
- caminando hacia la pelota,
- acomodándose para poder disparar o
- al estar disparando.

Estado 6: Levantarse. Este estado permite levantar al robot del suelo cuando se haya caído. El robot en cualquier momento puede caer. Por lo tanto cuando el robot se caiga, pasa a este estado, en el cual se implementó una nueva locomoción que no trae el robot por *default*: la locomoción de levantarse. En nuestra implementación se considera que para levantarse el robot tiene que estar acostado boca abajo. Si el robot está acostado boca arriba, se efectúa una locomoción para que ruede y quede acostado boca abajo (el robot real no puede levantarse boca arriba porque es demasiado el esfuerzo que hacen los servos de tal forma que se pueden dañar, por tal razón no se hizo la locomoción para que se levante boca arriba). Le toma aproximadamente 3 segundos al robot levantarse cuando ya se encuentra boca abajo.

Transición j. Levantado. Esta transición permite pasar del estado 6 al estado 1 después de haberse levantado. El robot finalmente pudo levantarse y tiene que volver al estado 1.

Capítulo 6.

Conclusiones, logros y trabajo a futuro

6.1. Conclusiones

En este proyecto de tesis se llevó a cabo el desarrollo de un simulador de comportamiento que permita jugar a un robot Nao en el juego de fútbol, integrando inteligencia artificial para que el robot pudiera jugar autónomamente. Esto permite que el robot tome decisiones para efectuar ciertas acciones en un momento dado

Los objetivos que se establecieron al principio de la tesis fueron cumplidos. El desarrollo de algoritmos de visión, de movimientos y el control de comportamientos fueron alcanzados satisfactoriamente.

Las contribuciones fueron alcanzadas a nivel simulador, pero por la ausencia del robot físico no se pudieron probar en el mundo real. La participación en el torneo de RoboCup en la categoría de simuladores se pretende alcanzar en el año 2011.

El Nao posee prestaciones muy buenas que lo convierten en una elección perfecta para ser utilizado en el simulador. Integra varios sensores así como un buen número de grados de libertad para una buena locomoción. El único inconveniente es el elevado costo de este robot, ya que por tal motivo no se pudieron implementar el comportamiento y comprobar que funciona correctamente en el mundo físico.

6.2. Logros

El objetivo general y los objetivos específicos descritos en el Capitulo 1 fueron alcanzados. En la locomoción cabe destacar los siguientes puntos:

- **Levantado**. Esta locomoción no existía en Webots y se implementó en esta tesis.

- **Caminado**. Esta locomoción está por *default* en el robot. En nuestro desarrollo logramos un caminado dinámico para que el robot caminara una cierta distancia sin la necesidad de detenerse constantemente.

- **Disparo normal**. Esta locomoción igual que la anterior ya existía, sin embargo se hicieron modificaciones para que el disparo normal de 1.3 metros originalmente alcanzara una distancia superior a los 2.2 metros y pudiera efectuar el tiro con ambas piernas.

- **Disparo lateral** (con pierna izquierda y derecha). Esta locomoción se desarrolló en esta tesis y se basó en el principio de tiro hacia adelante para poder efectuar tiros de costado.

- **Acomodamiento**. Para el acomodamiento se realizó una combinación de diferentes locomociones para que el robot pudiera rodear a la pelota sin la necesidad de tocarla, es decir, desplazarse a su alrededor para tener una mejor posición de disparo.

- **Rodar**. Esta locomoción no existía en Webots, por lo que se desarrolló en esta tesis. Esta locomoción se utiliza cuando el robot cae y está acostado boca arriba. En esta locomoción el robot debe "rodar" en el piso y ubicarse boca arriba para poder levantarse.

De la misma forma se mejoraron y se crearon nuevos algoritmos para el manejo de los sensores.

- **Búsqueda de portería**. El algoritmo desarrollado en esta tesis para la búsqueda de la pelota es más rápido que el que trae por *default* el simulador.

- **Búsqueda de pelota**. Se optimizó en un 90% el algoritmo original de Webots, el cual es más eficaz en la búsqueda.

- **Posicionar la pelota al centro**. Este algoritmo no existía anteriormente en Webots y fue desarrollado. Este algoritmo permite colocar la pelota al centro de la cámara.

- **Posición**. Se implementó un nuevo algoritmo para identificar si el robot está caído o parado con la ayuda del acelerómetro.

Se crearon los siguientes nuevos algoritmos para efectuar cálculos y decisiones más eficientes.

- **Cálculo de trayectoria**. Este algoritmo se creó para poder identificar la distancia y dirección en la que se encuentra la pelota a fin de llegar con mayor precisión.

- **Determinación de disparo**. Este algoritmo permite identificar con que pierna y hacia que dirección hay que efectuar el tiro.

6.3. Trabajo a futuro

El objetivo principal de esta tesis se cumplió, el cual consistía en lograr que un solo robot Nao, operando en un mundo virtual, pudiera realizar una serie de movimientos independientes a fin de localizar una pelota, dirigirla hacia una portería y disparar hacia la portería encontrada. Estas actividades, son la base para poder llevar a cabo la segunda fase de este proyecto que consiste en programar un conjunto de robots a fin de que jueguen al fútbol contra un adversario. Esta segunda fase del proyecto consistiría de las siguientes actividades genéricas:

- Programar dos robots para que realicen al mismo tiempo las actividades descritas en esta tesis.

- Programar la coordinación y comunicación entre robots a fin de que puedan coordinar jugadas.

- Programar las actividades del robot portero.

- Programar a tres Robots para que lleven a cabo el juego de fútbol contra un adversario.

Además, con base en la experiencia adquirida durante el desarrollo de esta tesis proponemos las siguientes actividades con el objetivo de mejorar los movimientos individuales de cada robot dentro de la cancha de juego.

- **Auto localizarse en el campo**. Esta actividad es muy importante para el juego de fútbol del robot, ya que le permitiría determinar en qué parte de la cancha se localiza y así poder diseñar estrategias de juego. Por ejemplo se

podría implementar una estrategia de juego en la cual el robot delantero no baje más de la media cancha y un defensa no suba más de la media cancha.

- **Comunicación con otro Nao**. En esta actividad se pretende lograr que el robot se comunique con otro u otros robots dentro de la cancha de juego.

- **Coordinación con otros Nao**. Una vez que un robot puede comunicarse con otro deben programarse distintas actividades para coordinar el juego entre 2 Robots. Por ejemplo, si el punto de la auto-localización es posible, entonces se puede determinar qué robot se encuentra más cerca de la pelota para que solo este vaya por ella, a fin de evitar que dos robots intenten realizar la misma actividad.

- **Mejoramiento de las locomociones**. En esta actividad se pretende lograr que el robot se desplacé más rápido en la cancha, que pueda lograr un caminado en curva y también realizar los tiros más velozmente.

Publicaciones del autor

[Ibarra et al., 2011] J.M. Ibarra Zannatha, L.E. Figueroa Medina, R. Cisneros Limón and P. Mejía Álvarez, 21st Internacional Conference on Electronics Communications and Computers CONIELECOMP 2011, IEEE Computer Society, 7 pages, Universidad de las Americas, Cholula, Puebla, Mexico, February 28 - March 2, 2011.

Anexo

Código para buscar la portería.

```
int NaoCam::findColorBlob2(const double ref[3], double tolerance,
double &direction) {
  int length = width * height/2;  // number of pixels in the image
  int izq1, izq2, der1, der2;
  int xmenor=160, xmayor=-1, i, bandera=0;
  int ymayor=59, entrada=0, diferencia, encontre=0;
  int npixels = 0, distancia=0, valor=0;
  double pixel[3];
  const unsigned char *p = image;
  p+=9440*3;
  for (i = ymayor*160; i < length; i++, p+=3) {
    rgbnorm(p, pixel);
    if (fabs(pixel[0] - ref[0]) < tolerance && fabs(pixel[1] -
ref[1]) < tolerance && fabs(pixel[2] - ref[2]) < tolerance) {
        if(xmenor > i%width)
           xmenor=i%width;
        if(xmayor < i%width){
           xmayor=i%width;
           if(bandera==2)
             der1=i%width;
           if(bandera==0 || bandera==2)
             bandera++;
        }
        npixels++;
    }
    else {
      if(bandera==1){
        bandera++;
        izq1=xmenor;
        izq2=xmayor;
      }
    }
  }
  if(bandera==0)
   return 0;
  if(bandera==3){
    posteizquierdo= (double((izq1+izq2)/2) / width - 0.5) * fov;
    postederecho=(double((der1+xmayor)/2) / width - 0.5) * fov;
    x1min=izq1;
```

```
    x1max=izq2;
    x2min=der1;
    x2max=xmayor;
    return 1;
  }
  if(bandera==2){
    p-=(i*3);
    distancia=(xmayor-xmenor)+2;
    if(xmenor > 0 && xmayor <159){
      for(i=0;i<59;i++){
        p+=(xmenor-1)*3;
        rgbnorm(p, pixel);
        if (fabs(pixel[0] - ref[0]) < tolerance && fabs(pixel[1] -
ref[1]) < tolerance && fabs(pixel[2] - ref[2]) < tolerance) {
         postederecho=(double((xmayor+xmenor)/2) / width - 0.5) *
fov;
         return 2;
        }
        p+=(distancia)*3;
        rgbnorm(p, pixel);
        if (fabs(pixel[0] - ref[0]) < tolerance && fabs(pixel[1] -
ref[1]) < tolerance && fabs(pixel[2] - ref[2]) < tolerance) {
         posteizquierdo= (double((xmayor+xmenor)/2) / width - 0.5)
* fov;
         return 3;
         }
        p+=(160-(xmayor+1))*3 ;
      }
      postedesconocido=((double((xmayor+xmenor)/2)-double(xmenor))
/ 1 / width - 0.5) * fov;
      return 4;
    }
    if(xmenor==0){
      for(i=0;i<59;i++){
        p+=(xmenor-1)*3+(distancia*3);
        valor+=(distancia)*3;
        rgbnorm(p, pixel);
        if (fabs(pixel[0] - ref[0]) < tolerance && fabs(pixel[1] -
ref[1]) < tolerance && fabs(pixel[2] - ref[2]) < tolerance) {
         posteizquierdo= (double((xmayor+xmenor)/2) / width - 0.5)
* fov;
         return 3;
         }
        p+=(160-(xmayor+1))*3 ;
      }
```

```
      postedesconocido=((double(xmayor)-double(xmenor))   /   1   /
width - 0.5) * fov;
      return 4;
    }
    else{
      for(i=0;i<59;i++){
        p+=(xmenor-1)*3;
        rgbnorm(p, pixel);
        if (fabs(pixel[0] - ref[0]) < tolerance && fabs(pixel[1] -
ref[1]) < tolerance && fabs(pixel[2] - ref[2]) < tolerance) {
         postederecho=(double((xmayor+xmenor)/2) / width - 0.5) *
fov;
         return 2;
        }
        p+=(distancia)*3+((160-(xmayor+1))*3);
      }
      postedesconocido=((double(xmayor)-double(xmenor))   /   1   /
width - 0.5) * fov;
      return 4;
    }
  }
}
```

Codigo para encontrar la pelota

```
void NaoCam::findColorBlob(const double ref[3], double tolerance,
double &direction, double &elevation) {

  int length = width * height;  // number of pixels in the image
  double x = 0, y = 0;
  int npixels = 0;
  const unsigned char *p = image;
  for (int i = 0; i < length; i+=2, p+=6) {
    double pixel[3];
    rgbnorm(p, pixel);
    if (fabs(pixel[0] - ref[0]) < tolerance && fabs(pixel[1] -
ref[1]) < tolerance && fabs(pixel[2] - ref[2]) < tolerance) {
      x += i % width;
      y += i / width;
      npixels++;
      //fout << "1";
    }
    if (i%160==0){
     i+=160;
     p+=480;
    }
  }
```

```
  if (npixels > 0) {
    direction = (x / npixels / width - 0.5) * fov;
    elevation = -(y / npixels / height - 0.5) * fov;
  }
  else {
    direction = UNKNOWN;
    elevation = UNKNOWN;
  }
}
```

Código de caminado dinámico

```
int FieldPlayer::caminaporlaPelota(double distancia){
 //si camara es la superior entonces
 int salir;
 double pasos, i;
 pasos=((distancia*2)/1.5)*8;     //1.354
 iniciocaminar();
 if(posicion()==0)
     return 0;
 for(i=0;i<pasos;i++){
    salir=intermediocaminar();
    if(salir==0)
     i=pasos;
  }
 finalcaminar();
 if(posicion()==0)
     return 0;
 return 0;
}
```

Referencias

[1] Cisneros, Rafael. Modelo matemático de un robot paralelo de seis grados de libertad. Trabajo de Tesis. Universidad de las Américas, Puebla, 2006.

[2] Cisneros, Rafael. Estrategias de Modelado Cinemático y Simulación en Robots Humanoides. Tesis de Maestría, CINVESTAV, México, DF, 2009.

[3] Siegwart, Roland y Nourbakhsh, Illah. Introduction to autonomous mobile robots. MIT Press, 2004.

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

[5] Carbone, Giuseppe y Ceccarelli, Marco. Legged robotic systems. Cutting Edge Robotics, 2005.

[6] Ibarra, Juan. Robots caminantes. Technical report, Departamento de Control Automático del CINVESTAV-IPN, 2003.

[7] Özyurt, Gökhan. 3-D humanoid gait simulation using an optimal predictive control.
Master's thesis, Middle East Technical University, 2005.

[8] RoboCup, http://www.robocup.org/, pagina consultada Enero 2010.

[9] Junco, Maria et al. RoboCup: El reto. Articulo. Tecnológico de Monterrey Campus Estado de México. Noviembre, 2009.

[10] Robostadium, http://robotstadium.org/, pagina consultada Enero 2010

[11] Aldebaran Robotics, http://www.aldebaran-robotics.com, pagina consultada Enero 2010

[12] Marc, Petter. Nao programming for the Robotstadium on-line contest. Master semester project, Swiss Federal Institute of Technology Lausanne (EPFL). Suiza, 2010.

[13] Certo, Giuseppe. Nao model and simulation for Webots. Bachelor Project. Swiss Federal Institute of Technology Lausanne (EPFL). Suiza, 2009.

[14] Cyberbotics. Webots User Guide. Suiza. 2009.

[15] Webots. www.cyberbotics.com pagina consultada Enero 2010.

[16] Cyberbotics. Webots Reference Manual. Suiza. 2009.

[17] Tellez, R. y Angulo C. Software review. Reporte técnico. Universidad Técnica de Catalonia. Barcelona, España. Agosto 2006.

[18] Olivier, Michel et al. Cyberbotics' Robot Curriculum. Wikibooks.Marzo 2009.

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

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

[21] Lewis, Harry. Elements of the theory of computacion. Second edition. Prentice-Hall. USA. 1998.

[22] Hopcroft, John y Ullman, Jeffrey. Introducción a la teoría de autómatas, lenguajes y computación. Primera edición. Editorial Continental. México. 1993.

[23] Zagal, J.C., del Solar, J.R.: UCHILSIM: A Dynamically and Visually Realistic Simulator for the RoboCup Four Legged League. In: RoboCup 2004: Robot Soccer World Cup VIII. Lecture Notes in Artificial Intelligence, Springer (2004).

[24] The Open Graphics Library. Available at http://www.opengl.org (2010).

[25] Tim Laue at el. SimRobot – A General Physicial Robot Simulator and its Application in RoboCup. Lecture Notes in Artificial Intelligence. Germany. 2005.

[26] Smith, R.: Open Dynamics Engine - ODE (2005) www.ode.org.

[27] B. Browning and E. Tryzelaar. ÜberSim: A Multi-Robot Simulator for Robot Soccer. Lecture Notes in Artificial Intelligence. USA. 2004.

[28] J. Wang. USARSim: A game-based simulation of the NIST reference arenas. School of Computer Science. Carnegie Mellon. USA. 2005.

[29] A. Howard y N. Koening. Gazebo Version 0.4.0 User Manual. USC Robotics Research Laboratory. University of Southern California. LA, California. USA .May 31, 2004

[30] Microsoft Robotics Studio. http://www.conscious-robots.com/es/robotics-studio/index.php. Consultada Mayo 2010

[31] J. Boedecker y M. Asada. SimSpark. Concepts and application in the RoboCup 3D soccer simulation league. Dept. of Adaptive Machine Systems. Osaka, Japan. 2008.

[32] Robonova I. http://www.superrobotica.com/Robonova.htm Pagina consultada Junio 2010.

[33] A. Borisov et al. NAO-Team Humboldt 2009. Institut für Informatik, LFG Künstliche Intelligenz, Humboldt-Universität zu Berlin, Germany .2008.

[34] P. Marc. Nao programming for the Robotstadium on-line contest. Ecole Polytechnique Fédérale de Lausanne EPFL. Suiza. 2010.

[35] K. Jahn et al. Nao-Team HTWK. Team Research Report 2009.Leipzig University of Applied Sciences. Germany. November, 2009.

[36] J. Pransky. AIBO. The No. 1 selling service robot. Industrial Robot: An International Journal. Volume 28. Number 1. 2001. Pag. 24-26.