UNIVERSIDAD REY JUAN CARLOS

Ingeniera Superior en Informática

Escuela Superior de Ciencias Experimentales y Tecnología

Curso académico 2009-2010

Proyecto Fin de Carrera

Componente BICA de coreografías para robot NAO aplicado a terapias en enfermos de Alzhéimer

Tutor: Francisco Martín Rico

Autor: Raúl Benítez Mejías

Madrid 2010

Este proyecto esta dedicado a mi mujer, Gema,
que ha sido un gran apoyo durante toda la realización
del proyecto.
A mis padres y hermana,
que también me han apoyado y me han animado
a seguir en los momentos más complicados.
Gracias.

Agradecimientos

Quiero dar las gracias a Francisco Martín y a Carlos Agüero por la ayuda recibida y el tiempo invertido en mi proyecto. También dar las gracias a los compañeros que me han ayudado en momentos de duda, en general a todos los miembros del Grupo de Robótica de la Universidad Rey Juan Carlos. De manera especial quiero dar las gracias a Francisco Martín, por todo el tiempo y conocimientos facilitados para la realización de este proyecto. Por último, agradecer a mi mujer, a toda mi familia y mis amigos por el apoyo ofrecido durante todo este tiempo.

Resumen

Este proyecto se encuadra dentro del campo de la Robótica, en particular, en el campo de la robótica móvil y dentro de éste, en la robótica asistencial. Es un proyecto en el cual se propone un sistema para poder mejorar y hacer más eficaces las sesiones de terapia en pacientes con enfermedades neurológicas.

El principal objetivo que se busca con la realización de este proyecto es conseguir un sistema con el cual se puedan generar de forma fácil y sencilla coreografías para un robot humanoide. Estas coreografías se puedan usar en las sesiones de terapia con enfermos de Alzhéimer.

Para la generación de las coreografías se dispondrá de una herramienta mediante la cual se podrán escribir coreografías con un lenguaje diseñado por nosotros. Además, desde esta herramienta se pondrán en marcha los guiones, se podrá avanzar y retroceder dentro de ellos de una forma ágil y sencilla.

Las coreografías se compondrán de las siguientes acciones: Realizar movimientos o bailes, reproducir ficheros de música, hablar, andar, juegos de luces, esperar a que acaben ciertas tareas o marcar zonas de avance y retroceso.

Cada una de estas acciones se pueden ejecutar en el robot por separado usando las aplicaciones correspondientes dentro de la herramienta general.

Este proyecto ha sido realizado en el Grupo de Robótica del departamento del GSyC de la Universidad Rey Juan Carlos.

Índice general

1. Introducción **9**

1.1. La robótica 10

1.2. Robótica Móvil 12

1.2.1. Problemas de la robótica móvil 13

1.3. Interacción Humano-Robot 14

1.4. Robótica asistencial 17

1.5. Robocup y su sentido 18

1.6. Grupo de Robótica de la URJC 22

1.7. Estructura de la memoria 24

2. Objetivos y Requisitos **25**

2.1. Objetivos 25

2.2. Requisitos 26

2.3. Método de desarrollo 27

2.3.1. Modelo en espiral 27

2.3.2. Prototipos del proyecto 28

3. Plataforma de desarrollo **30**

3.1. Nao 30

3.2. NaoQi 31

3.2.1. Coreografo Comercial 33

3.3. Bica(Behavior Based Architecture For Robot Applications) 34

3.3.1. VICODE: (VIsual COmponent DEsigner) 36
3.4. JManager 39
3.4.1. Depuración de los componentes 39

4. Descripción Informática **41**
4.1. Diseño del lenguaje de guiones 41
4.2. Desarrollo de los Componentes y de la interfaz dentro de JManager 45
4.2.1. Componentes 46
4.2.2. Herramienta para crear movimientos 60

5. Experimentos y Pruebas **65**
5.1. Pruebas unitarias 65
5.1.1. Music 65
5.1.2. TextSpeech 66
5.1.3. LedsControl 66
5.1.4. Body 67
5.1.5. Movie 67
5.1.6. Create Movie 68
5.2. Experimentos 69
5.2.1. Primera sesión 70
5.2.2. Segunda sesión 74

6. Conclusiones y Trabajos futuros **76**
6.1. Conclusiones 76
6.2. Posibles trabajos futuros 78

Bibliografía **79**

Índice de figuras

1.1. Brazo robótico de la empresa Festo y Robot cirujano 11
1.2. Asimo y Roomba 11
1.3. Mascotas: AIBO de Sony (a), iCat de Philips.(b) y BEAR, asistente de guerra (c) 12
1.4. Robot submarino y Robot aereo 13
1.5. Kismet 16
1.6. Valle Inexplicable 16
1.7. Twendy-One, Ri-man y Paro 17
1.8. Robot de rescate 19
1.9. Equipo de Complubot 20
1.10. Robot medianos, robot humanoide y robot de la plataforma estándar . . . 21
1.11. Grupo de Robótica 23

2.1. Método en espiral 28

3.1. Robot Nao de Aldebaran Robotics 31
3.2. Árbol de Brokers 31
3.3. Coreografor de Aldebaran Robotics 33
3.4. Creando un movimiento 34
3.5. Entradas y Salidas de un componente 35
3.6. Ejemplo 1 y Ejemplo 2 35
3.7. BICA, NaoQi, NAO y JManager 36
3.8. Máquina de Estados en VICODE 37

3.9. Componente Perception y Componente LpsViewer 40

4.1. Compente Movie dentro de BICA 46
4.2. Máquina de Estados Music 48
4.3. Componente Music 49
4.4. Máquina de Estados TextSpeech 50
4.5. Componente TextSpeech 51
4.6. Máquina de Estados Body 52
4.7. Componente Body 54
4.8. Máquina de Estados Led Control 55
4.9. Máquina de Estados Movie 56
4.10. Componente Movie 58
4.11. Interacción entre JManager y el Robot 59
4.12. Herramienta Create Movie 63
4.13. Motores de Nao 64

5.1. Terapia con robots 69

Capítulo 1

Introducción

Este proyecto propone un sistema para mejorar y conseguir mejores resultados en las sesiones de terapia con pacientes con enfermedades neurológicas. Para ello se va a contar con la ayuda de un robot humanoide. Por lo que se puede intuir, este proyecto estará encuadrado en el campo de la Robótica. Dentro de la robótica existen muchos campos que explorar. Este proyecto en concreto se centrará en la robótica móvil y dentro de éste en la robótica asistencial.

El sistema que se propone es un sistema basado en coreografías para que el robot humanoide los realice. El sistema tendrá un uso fácil e intuitivo para que cualquier persona, especialmente las terapeutas que llevan a cabo las terapias y aquellas que no estén relacionadas con la robótica, pueda utilizarlo sin necesidad de ser asistida por un técnico.

Las coreografías se podrán crear desde una herramienta. Estas coreografías se definirán mediante un lenguaje que hemos diseñado para ello. Gracias a este lenguaje, las coreografías se podrán crear de una forma fácil y rápida. Incluso se podrán modificar en cualquier momento de la sesión, dando así, una gran versatilidad a la aplicación. Por último, las coreografías se compondrán de acciones como bailes, reproducción de música, andar, hablar, luces, etc. . .

A continuación, vamos a describir como está estructurado este capítulo. En el punto 1.1 vamos explicar la historía de la Robótica, algunas definiciones de ésta y los tipos de robots que existen. En el punto 1.2 hablaremos de la robótica móvil, ya que es la parte de la robótica hacía la que está enfocada este proyecto. Explicaremos algunos de los problemas con los que nos encontramos al trabajar en el campo de la robótica móvil. En el punto 1.3 nos centraremos en una de las problemáticas, la interacción humano-robot, y veremos cuales

son estos problemas. En el punto 1.4 definiremos la robótica asistencial, en qué consiste y qué robots ya están haciendo estas funciones. En el punto 1.5 presentaremos uno de los bancos de pruebas más importantes que se está usando ahora mismo, la Robocup[1] y veremos qué nos aportan todos sus avances en la vida diaria. En el punto 1.6 hablaremos del grupo de robótica de la Universidad Rey Juan Carlos y, por último, en el punto 1.7 veremos como está organizada la memoria de este proyecto, para que sea más fácil encontrar los contenidos.

1.1. La robótica

A finales del siglo XX, principios de los años 70, surgió la posibilidad de crear objetos artificiales que actuasen de un modo autónomo. Estas se pueden considerar las primeras andaduras en el mundo de la robótica.

La palabra robot proviene de la palabra eslava *robota*[Capek, 1920] que se refiere al trabajo realizado de manera forzosa. Existen muchos trabajos que las personas no les gusta hacer, sea por aburrido o por peligroso. En la actualidad los robots son ideales para este tipo de trabajos, que requieren movimientos repetitivos, precisos o que son peligros como pueden ser la desactivación de bombas, limpieza, etc.

Existen múltiples definiciones de la palabra Robótica, entre las que destaco:

- **Real Académia Española** [2]. *Técnica que aplica la informática al diseño y empleo de aparatos que, en sustitución de personas, realizan operaciones o trabajos, por lo general en instalaciones industriales.*
- **Wikipedia** [3]. *Un robot es una entidad virtual o mecánica artificial. En la práctica, esto es por lo general un sistema electromecánico que, por su apariencia o sus movimientos, ofrece la sensación de tener un propósito propio. La palabra robot puede referirse tanto a mecanismos físicos como a sistemas virtuales de software, aunque suele aludirse a los segundos con el término de bots* [4].

La robótica es una disciplina con sus propios problemas, fundamentos y leyes. Tiene dos niveles: hardware y software. El software se refiere a la informática, a la programación,

[1] http://www.robocup.com
[2] http://www.rae.es
[3] http://es.wikipedia.org
[4] http://es.wikipedia.org/wiki/Bot

a la inteligencia artificial y a la automática. En el hardware hay aspectos de mecánica, electrónica, de control y de construcción.

Desde un punto de vista muy general los robots podrían clasificarse de la siguiente manera:

1. **Fijos**. Son artilugios mecánicos que se encargan de realizar de forma automática procesos de fabricación o manipulación. Suelen tener forma de brazo articulado, en cuyo extremo incorporan elementos de sujección o herramientas. Realizan tareas repetitivas en industrias de automoción, fabricación mecánica o electrónica en las que se emplean para montar y mover piezas o componentes, soldar, pintar, etc. . . Destacar que el campo de acción de los robots fijos está limitado a la distancia que mida el brazo del robot.

(a) (b)

Figura 1.1: Brazo robótico de la empresa Festo y Robot cirujano

2. **Móviles**. Son los robots que están provistos de patas o ruedas que son capaces de moverse por su entorno según su programación. Elaboran la información que reciben a través de sus propios sensores y se desplazan en función a ella. Estos robots también se utilizan para exploración espacial (como pueden ser *Spirit* u *Opportunity*), rescates en lugares de díficil acceso, etc. . .

(a) (b)

Figura 1.2: Asimo y Roomba

Este proyecto, dentro de la clasificación que hemos hecho anteriormente, hay que decir que pertenece a la robótica móvil. Dentro de la robótica móvil, hay diversos campos de investigación, en concreto, este proyecto está enfocado hacía el campo de la robótica asistencial, de la cual hablaremos más adelante, en la sección 1.4.

1.2. Robótica Móvil

Cada uno de los robots que tiene capacidad de desplazamiento mediante algún tipo de sistema locomotor, como puedan ser, las ruedas, unas piernas, hélices, etc... y que, además son capaces de realizar estos desplazamientos gracias a la información de su entorno mediante el uso de sus sensores, se pueden considerar robots que pertenecen al campo de la robótica móvil.

Existen diferentes clasificaciones atendiendo a la forma de moverse y según cómo se muevan. Pueden existir robots *teleoperados*, es decir, los maneja un ser humano, y los que son *autónomos*, toman decisiones según lo necesiten.

Además, también los podemos clasificar según su forma de moverse, algunos vuelan, otros se mueven gracias a que tienen patas, otros piernas, otros ruedas, otros son capaces de moverse por el agua, etc...

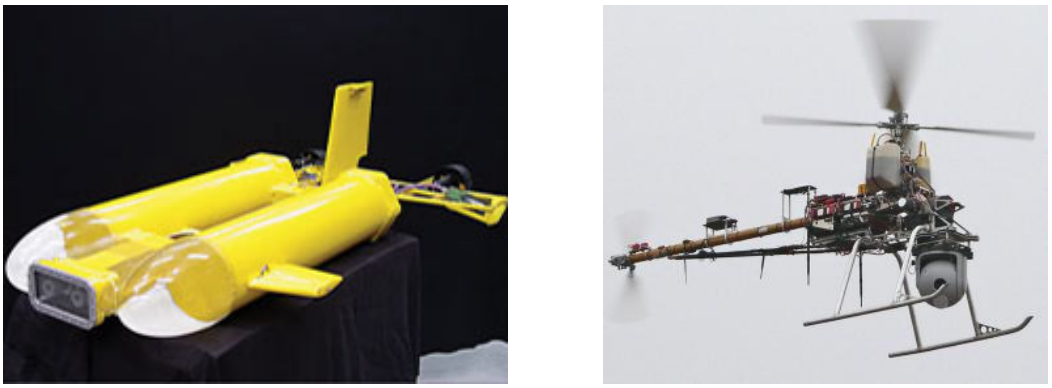
(a) (b) (c)

Figura 1.3: Mascotas: AIBO de Sony (a), iCat de Philips.(b) y BEAR, asistente de guerra (c)

Aparte de la investigación de la robótica móvil para crear máquinas que realicen trabajo, también se está investigando la robótica orientada al ocio y a la ayuda a los seres humanos.

Existen robots en museos, robots domésticos que limpian la casa, que presentan las noticias, reproducen música o cuidan de nuestras mascotas. Estas nuevas aplicaciones para los robots hacen que los problemas de autonomía aumenten. Algunos de estos problemas son la navegación, la localización, la interacción humano-robot, etc. . . de los que hablaremos en el siguiente punto.

Algunos de los diferentes robots y sus diferentes funciones se pueden observar en las figuras 1.3 y 1.4.



(a) (b)

Figura 1.4: Robot submarino y Robot aereo

1.2.1. Problemas de la robótica móvil

En este apartado vamos ha comentar algunos de los problemas clásicos de la robótica móvil. Como hemos comentado en el apartado anterior, estos problemas, por el hecho de desplazarse por su entorno, son la navegación, la localización, la percepción o la interacción humano-robot.

- **Percepción**. Obtener información nítida a través de los sensores como el láser, los ultrasonidos, las cámaras, etc. . . es díficil ya que siempre se obtiene alguna información que es erronea. Obtener información de los sensores es una tarea clave en el desarrollo del software de los robots móviles.
- **Localización**. Podemos distinguir dos problemas.

  1. Conocer la posición local del robot conociendo la posición inicial, donde habrá que tener en cuenta los errores acumulados del sistema locomotor, como pueden por los odómetros.

2. Conocer la posición global dentro de un entorno sin conocer la posición inicial, en este caso aumentan los factores de error.

- **Navegación**. Es la capacidad que tiene un robot para poder ir de un punto conocido a otro. Depende mucho de los dos problemas que hemos comentado anteriormente ya que para poder navegar con fiabilidad, tenemos primero que estar bien localizados en el entorno en el que nos estamos moviendo y segundo tenemos que obtener una información válida de lo que estamos viendo.

- **Intereracción humano-robot**. La comunicación entre los humanos y los robots es el principal problema que abordamos en este proyecto. En el siguiente apartado se explicará más a fondo este problema.

Como se puede observar muchos de los problemas de la robótica móvil están ligados entre sí, y la resolución de unos influyen en los otros.

1.3. Interacción Humano-Robot

La interacción humano-robot es el estudio de las interacciones que existen entre humanos y robots. Los investigadores los denominan HRI (*Human Robot Iteration*). La interacción humano-robot es un campo multidisciplinar que abarca la psicología, inteligencia artificial, robótica, comprensión del lenguaje, etc...

La interacción humano-robot ha sido un tema de ciencia ficción y la especulación académica ha existido incluso antes de que existiesen los robots. El origen de HRI como un problema discreto empezó con la novela *Yo, Robot*[Asimov, 1950], escrita por Isaac Asimov. En esta novela se describen las tres reglas de la robótica, orientadas a regular la relación de convivencia entre humanos y robots:

1. *Un robot no debe dañar a un ser humano o permitir que un humano sufra ningún daño.*

2. *Un robot debe obedecer las órdenes dadas por los seres humanos, excepto cuando tales órdenes entren en conflicto con la primera ley.*

3. *Un robot debe proteger su propia existencia mientras dicha protección no entre en conflicto con la primera o segunda ley.*

Estas tres leyes determinan que exista una interacción muy fuerte. Cuando más cerca este el humano del robot, más posibilidades de que se causen daños. En la industria, esto se soluciona, separando el espacio de trabajo de los robots y de los humanos. De esta manera, la presencia del humano estará prohibida en una zona donde el robot esté trabajando. Gracias a los avances en la inteligencia artificial, los robots autónomos podrán llegar a tener mejores comportamientos, gracias a los que podrán planificar mejor sus movimientos en entornos desconocidos para ellos. Gracias a estas nuevas capacidades se podrán evitar que se incumplan la primera y la segunda ley. La investigación para llegar a este punto son la detección de humanos, planificación de movimientos, reconstrucción de la escena, plafinicación de tareas, etc...

El uso de los robots, aunque es más generalizado en la industria, se tiende a que se vayan incluyendo en labores de búsqueda y rescate, combate militar, detección y desactivación de minas, entretenimiento y atención hospitalaria.

La investigación de HRI abarca varios aspectos. Uno de ellos es la detección de humanos. Para ello existen varios métodos, la mayoría ellos tienen la intención de construir un modelo 3D a través de la visión del robot. Otros son el uso de los sensores que también nos sirven para recoger información a partir de una referencia.

La interacción humano-robot es un área que está aún poco explotada, aunque existe un desarrollo activo y constante en áreas como la movilidad, la navegación, la planificación, control de funcionamiento y todo ésto para poder conseguir aplicaciones para la eduación, el entretenimineto, la asistencia hospitalaria, la rehabilitación o el cuidado de ancianos.

Uno de los primeros robots que se creo en el que se puede ver la interacción humano-robot es con el robot *Kismet*, mostrado en la figura 1.5. Se trata de un robot que se creó con sistemas de expresión, de visión y de audición para poder interactuar con los humanos y simular la emoción y la apariencia humana.

Kismet consigue simular la emoción a través de las expresiones faciales, la vocalización y el movimiento. Parte de esto, lo consigue gracias a las cuatro cámaras que tiene montadas en la cabeza, que le permiten decidir a qué prestar atención y a qué no. El comportamiento visual lo puede realizar gracias a que tiene tres grados de libertad en los ojos, lo que le permite poder mover y orientar sus ojos como un ser humano. Además tiene 12 grados de libertad más, que le permite tener una gran variedad de expersiones faciales para expersar su estado emocional como, por ejemplo, poder guiñar un ojo, poner una sonrisa, poner el ceño fruncido, mostrar enfado, tristeza, etc...

Por último, la aceptación que tienen los humanos hacía los robots es un tema complejo.

Figura 1.5: Kismet

Vamos a explicar una hipótesis llamada *El Valle Inexplicable* de Masahiro Mori. Esta hipótesis dice que: *cuanto más se parece un robot a un humano tanto en su apariencia como en sus movimientos más empatía genera en los humanos, pero llegado a un punto el robot genera repulsión.* Se puede ver en la siguiente figura 1.6. Algunos robots como *Paro*, al no tener una apareciencia humana tienen una gran aceptación, en cambio, otros robots que tienen, por ejemplo la cara muy parecida a la de los humanos si obtienen ese rechazo.

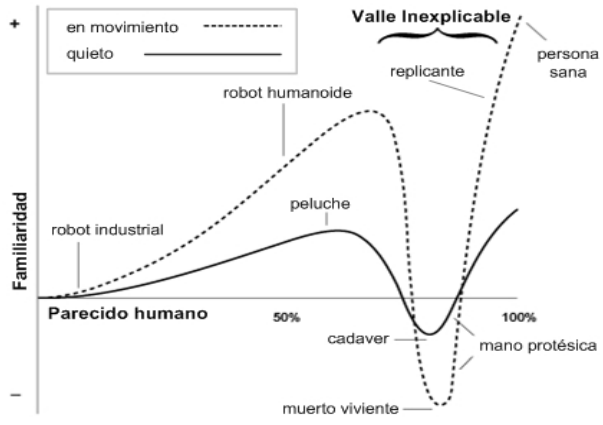
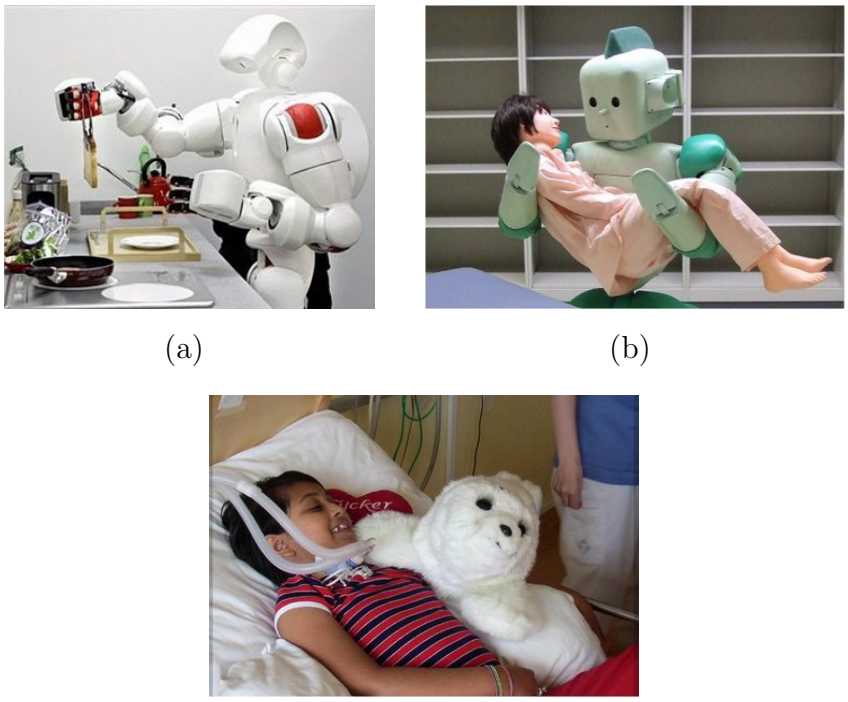


Figura 1.6: Valle Inexplicable

Que un robot se parezca mucho a un humano pero que tenga ciertos defectos, hace que lo rechacemos ya que somos reacios a relacionarnos con seres que tienen aparciencia enferma.

1.4. Robótica asistencial

La esperanza de vida cada vez es mayor. Esto puede llegar a implicar que cada vez habrá más gente anciana y ésta puede que necesiten ser atendidos por alguien o por *algo*. Hasta ahora siempre eran otras personas las que atendían a las personas mayores pero vamos encaminados a que sean los robots quienes puedan realizar esta trabajo. Ya existen sistemas que son capaces de monitorizar a ancianos y que detectan cuando se sientan, cuando duermen, cuando se caen y registran su tensión arterial o su glucemia. Si alguno de estos aparatos detecta algo raro, los familiares pueden ser avisados inmediatamente. También existen robots que les recuerdan la hora a la que hay que tomarse la medicación.



(a) (b)

(c)

Figura 1.7: Twendy-One, Ri-man y Paro

En general, los robots son capaces de desarrollar una interacción especial (luces, sonidos, movimientos...). Por ejemplo, el robot *Tewndy-One*, en la figura 1.7(a), es un robot asistencial que lo mismo te prepara un sandwich que ayuda a levantarse al abuelo. El robot *Ri-man*, en la figura 1.7(b), tiene como función poder llevar a una pesona de un punto a otro, como si de un ser humano se tratase. Por último podemos hablar de *Paro* Robot, en la figura 1.7(c), que es una foca de peluche robotizada que anima y acompaña emocionalmente a ancianos con demencia o niños hospitalizados.

Gracias a varios estudios se ha detectado que la interacción de la gente con los animales de compañia les sirve para relajarse, aliviar el estrés, para la rehabilitación física, etc... Estos son los objetivos por lo cual se utilizan animales de compañia para las terapias.

El problema de los animales de compañía, es que, en los hospitales no se aceptan por los problemas de alergia, infecciones, y demás problemas que se puedan causar. Estos son algunos de los motivos, por lo cual se creó el robot *Paro*.

Paro ha sido una de las inspiraciones para la realización de este proyecto con Nao. Ya que, gracias a que Nao tiene un aspecto encantador y es como un *humano*, podemos realizar más actividades con las personas y podremos interactuar más con él. Mientras que usando *Paro* sólo podemos acariciarlo y hablar con él. Con *Nao* podremos realizar bailes mientras nos reproduce algún tipo de música. Podremos ver los estímulos de las personas cuando realiza juegos de luces con sus leds, además de poder hablar con él y que nos responda. La idea es que gracias a su forma, el que tenga más sensores y más capacidad de movimiento, nos dará una mayor libertad para realizar mejores terapias para los enfermos.

1.5. Robocup y su sentido

El Grupo de Robótica, del cual hablaremos en el punto 1.6, lleva muchos años investigando y participando en la Robocup. Gracias a las investigaciones y las pruebas que se han llevado a cabo, se han encontrado muchas de las soluciones que se proponen en este proyecto. El objetivo final de la participación en estas competiciones, a parte de que los robots jueguen al fútbol, es encontrar y poder probar nuevas soluciones ante los problemas que hemos mencionado antes que tiene la robótica móvil.

RoboCup [5] es un proyecto internacional para promover, a través de la competición de robots autónomos, la investigación y educación sobre inteligencia artificial. Esta competición mundial es uno de los eventos más conocidos en robótica móvil. Nació en 1996 como una iniciativa para fomentar la ciencia y la tecnología proponiendo una plataforma común de experimentación. Cada año se organizan campeonatos en diversas categorías (simulador, robots pequeños, robots medianos, perritos de Sony, Nao de Aldebaran Robotics).

Presenta un entorno dinámico, de tiempo real y distribuído. Por ello supone un escenario desafiante para la investigación en robótica, agentes, inteligencia artificial, etc... con el aliciente de la competición y la vistosidad.

[5] http://es.wikipedia.org/wiki/Robocup

La RoboCup está dividida en cuatro competiciones: *RoboCupSoccer, RoboCupRescue, RoboCupJunior, RoboCup@Home*. Y cada una de ellas, tienen sus propias ligas, dependiendo de cada modalidad.

Las útimas ediciones de la Robocup han sido celebradas en Atlanta, E.E.U.U (2007), Shuzhou (2008), siendo la última en Austria (2010). Este año 2010, la celebración se llevará a cabo en Singapur.

- **RoboCupRescue** Se trata de poner a prueba a los robots en tareas de búsqueda y salvamento de víctimas en terreno desfavorable.

  Los robots pueden ser tanto autónomos como guíados por control remoto. En cada prueba, un equipo de robots debe ser capaz de encontrar el camino adecuado en la zona designada y hacer un mapa con la posición de los diferentes obstáculos del térreno y de las víctimas.

  Hay dos ligas diferentes, en función de si se lleva a cabo en el terreno físico o virtual: la liga de simulación de rescate y la liga de rescate real.

Figura 1.8: Robot de rescate

- **RoboCupJunior** En esta competición, lo que se intenta es acercar las metas y objetivos de Robocup a estudiantes de educación primaria y secundaria.

  Hay diferentes formatos: fútbol, rescate y baile.

  En la prueba del fútbol los estudiantes, como el equipo de Complubot[6] en la figura 1.9, tienen que diseñar, construir y programar dos robots que sean capaces de jugar al fútbol contro otros dos robots. El campo en el que se juega tiene un tamaño

[6]http://complubot.educa.madrid.org

aproximado de una mesa de ping-pong sobre el que hay un degradado de grises para ayudar a posicionar a los robots. Las paredes son altas de forma que impiden que ningún robot se pueda salir del campo. Para poder localizar con facilidad la pelota esta emite luz infraroja.

-

Figura 1.9: Equipo de Complubot

En la prueba de rescate tienen que localizar *víctimas* en un escenario tridimensional que simula una catastrofe. El robot deberá seguir una línea, que no está siempre, y evitar obstaculos y salvar algunas rampas.

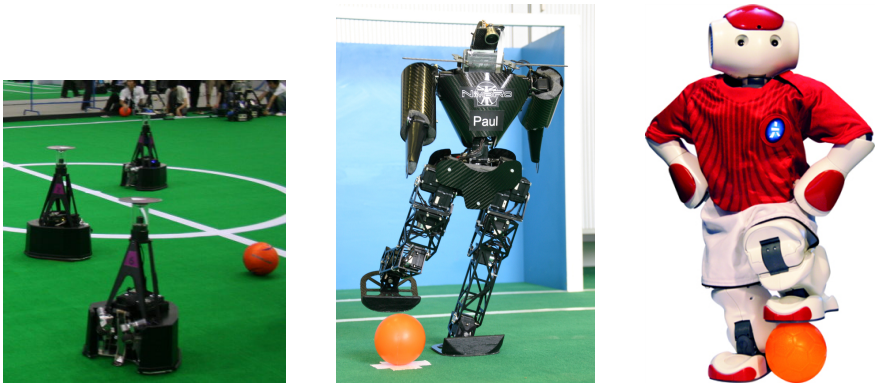
En la prueba del baile el objetivo es integrar la ciencia, la tecnología y el arte. Para ello los robots que participen en esta prueba se tienen que mover, de forma armoniosa, al ritmo de la música. Los participantes, además de diseñar y construir los robots, tendrán que hacer los decorados, vestuarios e incluso formar parte activa de la misma representación.

- **RoboCup@Home** Es la competición que se centra en las aplicaciones del robot para la vida diaria y las relaciones humano-robot. El escenario de la competición suele ser la vida real, que es donde de verdad se ven las capacidades de los robot.

- **RoboCupSoccer** Por último, esta competición, en la que trabaja el grupo de robótica, consiste en realizar partidos de fútbol con robots autónomos. El objetivo de esta competición es conseguir un equipo de fútbol de robots que sea capaz de jugar con humanos. Existen diferentes ligas, según la morfología del robot, y cada una tiene sus propias reglas.Existen varías ligas, que son:

  - **Simulación**. Son agentes software que juegan en un campo virtual dentro de un ordenador. Esta liga es una de las más antiguas en la RoboCup. Exiten varias subligas dentro de ésta: 2D, 3D...

- **Robots de tamaño pequeño**. En esta liga, los robots no pueden medir más de 18 cm de diametro, no son autónomos y usan una cámara cenital para moverse. Podrán jugar en equipos de hasta 5 robots con una pelota de color naranja en un campo de 6.5x4.5 metros. Los partidos se dividen en dos mitades de 10 minutos. El objetivo de esta liga es conseguir la cooperación multi-agente mediante sistemas distribuidos o centralizados.
- **Robots tamaño medio**. En esta liga, los robots no pueden medir más de 50 cm de diametro, son autónomos y llevan ruedas. Podrán jugar en equipos de hasta 6 robots con una pelota de color naranja en un campo de 12x18 metros. Los partidos se dividen en dos mitades de 15 minutos. Los sensores que use el robot tendrán que estar en el robot y podrán utilizar redes inalámbricas para comunicarse entre ellos. En la figura 1.10(a) tenemos un ejemplo.



(a) (b) (c)

Figura 1.10: Robot medianos, robot humanoide y robot de la plataforma estándar

- **Robots Humanoides**. Esta liga fue creada en el 2002 y está centrada en la construcción de los robots. Los robots humanoides tienen dos piernas y son autónomos para poder jugar partidos. Esta liga tiene dos subcategorías, según el tamaño. Existen grandes desafios técnicos, ya que, los robots tienen que caminar de una forma dinámica, correr, golpear la pelota mientras mantienen el equilibrio, tienen una percepción visual de la pelota, del resto de los jugadores y tienen que saber donde están en el campo, es decir, la autolocalización. Otro de los aspectos técnicos que se investigan es el juego en equipo. En la figura 1.10(b) tenemos un ejemplo.
- **Plataforma estándar**. Esta plataforma de la liga reemplaza a la liga de robots de cuatro patas. Todos los participantes tienen el mismo robot, (figura: 1.10(c))

y lo que se busca, es que toda la investigación se centre en el desarrollo del software. En esta liga los robots son autónomos, bipedos, usan la visión como principal sensor, aunque es una liga joven es la más atractiva. Esta es la liga en la que el Grupo de Robótica participa todos los años.

El objetivo de participar en la Robocup no es realmente jugar al fútbol sino investigar para conseguir ventajas para la sociedad. La Robocup en algunas de sus ligas, es un escenario que le permite a los estudiantes desde que son pequeños, interesarse por la ciencia; permitiéndoles aprender de una forma más práctica, sencilla y motivadora, donde se logra que ellos sean creadores e investigadores y no sólo consumidores de conocimientos y de productos. Esto hará a la larga que esos estudiantes esten interesados por la investigación en un fúturo.

En uno de los campos donde más beneficios se pueden obtener de la robótica es en la medicina [7], donde podemos encontrar robots cirujanos que tienen siempre la misma precisión y no sufren cansancio o fatiga. Además de la construcción de brazos, piernas o implantes para que alguien pueda ver o pueda volver a andar. Todas estas mejoras provienen de los campos de pruebas y de las investigaciones en la robótica.

Gracias a que desde el departamento de robótica de la Universidad Rey Juan Carlos están implicados en desarrollar tecnología para poder participar en la Robocup todos los años, se ha podido utilizar las investigaciones ya realizadas para realizar este proyecto, que será un beneficio directo para la sociedad, desde el punto de vista que servirá para mejorar las sesiones y hacerlas más efectivas con los enfermos. Sólo por proyectos como este o futuras aplicaciones que puedan salir de ahí, merece la pena seguir investigando y participando en esta competición.

1.6. Grupo de Robótica de la URJC

El grupo de Robótica[8] de la Universidad Rey Juan Carlos está formado por profesores y alumnos. El interés de este grupo está basado en la **generación de comportamiento artifical en robots**, tocando multitud de temas como *lógica borrosa, visión artificial, estimación, teoría de control, elaboración de mapas, inteligencia artificial, arquitecturas de control, agentes...*

El grupo (figura 1.11) dispone de:

[7] http://www.articuloz.com/tecnologia-articulos/los-robots-ante-la-sociedad-1518954.html
[8] http://www.robotica-urjc.es

- 30 robots lego
- 6 robots EyeBot
- 2 robots Pionier
- 9 robots Aibo de Sony
- 2 robots Nao Aldebaran Robotics

Figura 1.11: Grupo de Robótica

El grupo de Robótica lleva tiempo montando equipos capaces de jugar al fútbol y competir en la RoboCup[9]. Los equipos que se han montando han participado en las ligas de robots pequeños, de robots de cuatro patas y en la que está participando ahora mismo, que es en la plataforma estándar.

Otra de las líneas del grupo es la programación de robots para que exhiban un amplio repertorio de *comportamientos autónomos en entornos de oficina*. Para ello han desarrollado una arquitectura llamada JDE(Jerarquía Dinámica de Esquemas)[10].

[9]http://www.robocup.org
[10]http://www.jderobot.org

1.7. Estructura de la memoria

En este apartado vamos a describir como está estructurada la memoria.

En el capítulo 2 hablaremos de los objetivos que se buscan con la realización de este proyecto, los requisitos que son necesarios para que se cumpla lo que estamos buscando y por último, explicaremos el método de desarrollo que hemos seguido en este proyecto.

En el capítulo 3 vamos a hablar sobre la plataforma de desarrollo que hemos usado para realizar este proyecto. Contaremos en qué consiste y cómo funciona NaoQi, que es la plataforma usada para realizar este proyecto, la arquitectura BICA y la herramienta JManager. También hablaremos del robot que hemos usado para realizar este proyecto, Nao.

En el capítulo 4 haremos una descripción informática en la que explicaremos la herramienta principal para el usuario y haremos una descripción detallada de los componentes desarrollados y, por último, explicaremos el lenguaje para desarrollar las coreografías.

En el capítulo 5 explicaremos las pruebas y experimentos que se han realizado durante y tras finalizar el proyecto. También haremos referencia a los experimientos realizados en el hospital con los enfermos de Alzhéimer.

Por último, en el capítulo 6 veremos las conclusiones que hemos obtenido tras realizar el proyecto y las mejoras que podría añadirse.

Capítulo 2

Objetivos y Requisitos

En este capítulo vamos a describir los objetivos de este proyecto y los requisitos de los que partimos. También explicaremos el método de desarrollo que hemos seguido que es un modelo en espiral.

El objetivo principal de este proyecto es la creación de un sistema lo más sencillo para el usuario posible, para crear coreografías que se puedan aplicar en las terapias con enfermos de Alzhéimer. Asímismo, se explicarán los requisitos que serán necesarios para asegurar un buen comportamiento.

2.1. Objetivos

El principal objetivo que se busca con este proyecto es realizar un sistema que nos sirva para generar coreografías de una forma fácil, que son interpretadas por el robot Nao. Este sistema deberá seguir una arquitectura software, que será explicada más adelante.

La idea de realizar este proyecto viene dada para poder realizar terapias en hospitales con personas que sufren enfermedades neurológicas. Gracias a estas coreografías se consiguen estímulos sobre los enfermos que ayudan a su mejoría. Por ello, hay que desarrollar una interfaz sencilla e intuitiva, para que cualquier persona pueda crear las coreografías para el robot y llevar a cabo estas terapias.

Antes de implementar esta funcionalidad habrá que desarrollar otras funcionalidades más simples como, por ejemplo, reproducir ficheros de música, hablar, realizar los movimientos, etc... Estas funcionalidades más simples, nos permitirán desde la funcionalidad principal, modularlas y recoger información, para así poder reproducir las coreografías escritas.

Como se ha comentado, habrá que desarrollar primero varias funcionalidades:

- `Body`. Se encargará de realizar los movimientos en el robot.
- `TextSpeech`. Se encargará de reproducir de forma hablada, los textos.
- `Music`. Se encargará de reproducir los ficheros de música.
- `LedsControl`. Se encargará de la iluminación de los leds del robot.

Otro de los objetivos de este proyecto es diseñar un lenguaje que sea fácil e intuitivo para la generación de las coreografías. Este lenguaje estará basado en palabras en inglés relacionadas con la función que tendrá que realizar el robot.

El objetivo es ampliar la arquitectura BICA, con la cual vamos a desarrollar las funcionalidades que antes hemos comentado. Se aprendrá y ampliará de nuevas funcionalidades la aplicación JManager y se conocerá el funcionamiento de NaoQi. Estos tres conceptos, serán explicados más adelante.

2.2. Requisitos

El desarrollo del proyecto tiene los objetivos ya explicados anteriormente y se deberá ajustar a los siguientes requisitos para asegurar su buen comportamiento en el robot:

- El comportamiento deberá funcionar y realizar lo que se ha propuesto al inicio del proyecto, que es realizar un sistema para generar coreografías que pueda ejecutarlas el robot.
- El software se ha de desarrollar sobre la plataforma NaoQi, explicado en la sección 3.2.
- Se deberá usar la arquitectura BICA para el desarrollo de los componentes, presentado en la sección 3.3.
- El interfaz que se deberá usar y ampliar será el que usa el grupo de robótica, JManager, se explicará en la sección 3.4.
- El comportamiento debe poder ejecutarse en el robot Nao, versión 1.6, descrito en la sección 3.1.

- Los lenguajes de programación que se deben utilizar son C++, para el desarrollo sobre NaoQi y Java, para el desarrollo en JManager.
- Para poder ejecutar JManager, se podrá utilizar un sistema Linux o un sistema MacOS, debido a que es un sistema multiplataforma. Y para poder ejecutar el comportamiento, que se deberá poder ejecutar tanto en remoto desde un ordenador usando Linux, como desde el robot Nao. En el robot Nao, se podrá ejecutar como módulo dentro de él o como un ejecutable. Esto se explicará más adelante.
- Con la interfaz de usuario JManager, se debe poder interactuar con los comportamientos desarrollados.
- El sistema operativo será GNU/Linux, Ubuntu 2.6.

2.3. Método de desarrollo

En este apartado vamos a explicar el método de desarrollo en el cual nos hemos basado para la realización de este proyecto.

La metodología usada para este proyecto está basada en el modelo de espiral que se basa en prototipos. Este modelo es muy flexible a la hora de añadir o quitar requisitos, ya que, se hacen bastantes iteraciones, y al finalizar cada prototipo se pueden modificar los requisitos si es necesario.

Se ha escogido este modelo de desarrollo para hacer que el comportamiento final sea una fusión de comportamientos más pequeños y simples. Cada comportamiento tendrá sus propios requisitos que deberán cumplir para que la fusión sea buena.

2.3.1. Modelo en espiral

Como se acaba de comentar en el apartado anterior, al usar el modelo en espiral, el desarrollo del proyecto se genera gracias al número de iteraciones.

Las iteraciones que hay que realizar son las siguientes:

- **Determinar los objetivos**. Los objetivos de un ciclo de desarrollo deben de ser identificados y especificados, en nuestro caso, el objetivo es conseguir un comportamiento para generar una coreografía.

- **Valorar y reducir los riesgos**. Tendremos en cuenta los posibles fallos que pueda tener comportamiento.
- **Desarrollar y validar**. El sistema se desarrolla y es validado usando pruebas que verifican el cumplimiento de los requisitos fijados. En nuestro caso, haremos las pruebas oportunas para cada uno de los prototipos que se mencionarán más adelante.
- **Planificar**. El proyecto es repasado y la próxima fase de la espiral es planificada, en nuestro caso, una vez acabado y verificado el prototipo actual, se pasa a plafinicar el siguiente prototipo.

En la figura 2.1, se puede observar las cuatro etapas que forman este módelo de desarrollo: Análisis de requisitos, diseño e implementación, pruebas y planificación del próximo ciclo de desarrollo.

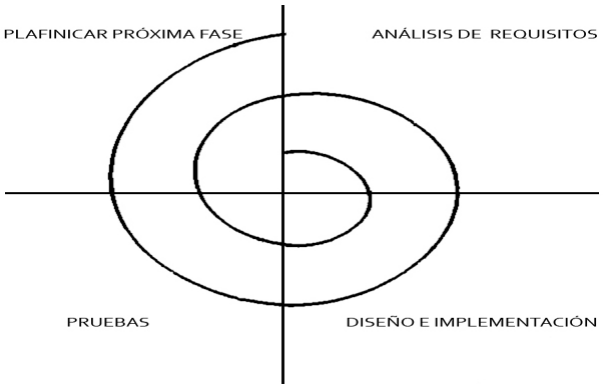


Figura 2.1: Método en espiral

2.3.2. Prototipos del proyecto

Un prototipo es una versión preliminar de un sistema con fines de demostración o evaluación de ciertos criterios. Se suele estimar la finalización de una iteración en el modelo de desarrollo como la obtención de un nuevo prototipo. El modelo en el cual nos estamos basando para la realización de este proyecto se basa justamente en esto, después de cada iteración, en conseguir un nuevo prototipo para poder realizar las pruebas oportunas.

Los prototipos que hemos realizado durante la realización del proyecto han sido:

- Prototipo 1: Diseño del interfaz de usuario de los compomentes. En este prototipo de lo que se trata es de establecer cual es el interfaz necesario para el control de los

diferentes componentes que se van a desarrollar. Esto nos llevará a añadir nuevos botones y pestañas a la herramienta JManager (para poder activar y desactivar los componentes por separado y el componente final en general).

- Prototipo 2: Desarrollar los componentes. En este prototipo la idea es desarrollar cada uno y por separado de los componentes, esto hará que existan varias iteraciones entre el prototipo 1 y el 2. Ya que, cada uno llevará consigo el desarrollo de la interfaz y el desarrollo del componente.

- Prototipo 3: Desarrollo del componente final. La idea de este prototipo, es desarrollar el componente final, en el cual se desarrollará primero la interfaz de usuario, y por último el desarrollo del componente y la realización de la fusión del resto de componentes. Esto nos permitirá poder utilizar todos los componentes desde éste último y realizar la coreografía buscada.

Cada prototipo mostrado tendrá sus requisitos, su diseño e implementación.

Capítulo 3

Plataforma de desarrollo

En este capítulo vamos a describir la plataforma de desarrollo usada para realizar este proyecto, la arquitectura y la herramienta de uso. Además veremos también el robot con que hemos trabajado.

Los elmentos más importantes son: **Nao** (sección 3.1), **NaoQi** (sección 3.2), **BICA** (sección 3.3) y **JManager** (sección 3.4).

Para el desarrollo de los componentes se ha utilizado el lenguaje de programación C++, tal y como está propuesto en los requisitos. Para la ampliación de la interfaz JManager se ha utilizado Java, también propuesto en los requisitos.

3.1. Nao

El robot Nao es un robot humanoide completamente programable. Esta equipado con: CPU AMD 500 Mhz x86 Geode, memoria Flash de 1 GB, 256 MB SDRAM, 2 altavoces, 2 camaras (no se pueden usar en estereo), conexión Wifi, tarjeta de Red, 25 grados de libertad, Linux 2.6 y un microcontrolador ARM 7 en el pecho para controlar los motores del robot y los sensores, llamados DCM.

El robot tiene algunas limitaciones, como es un microprocesador no muy potente y una memoria muy limitada. Gracias a sus numerosos grados de libertad nos permite realizar muchos movimientos. Por otro lado, tiene 2 cámaras de las cuales podemos obtener mucha información, por lo que al final es un robot perfecto para la investigación y la realización de diferentes tipos de proyectos.

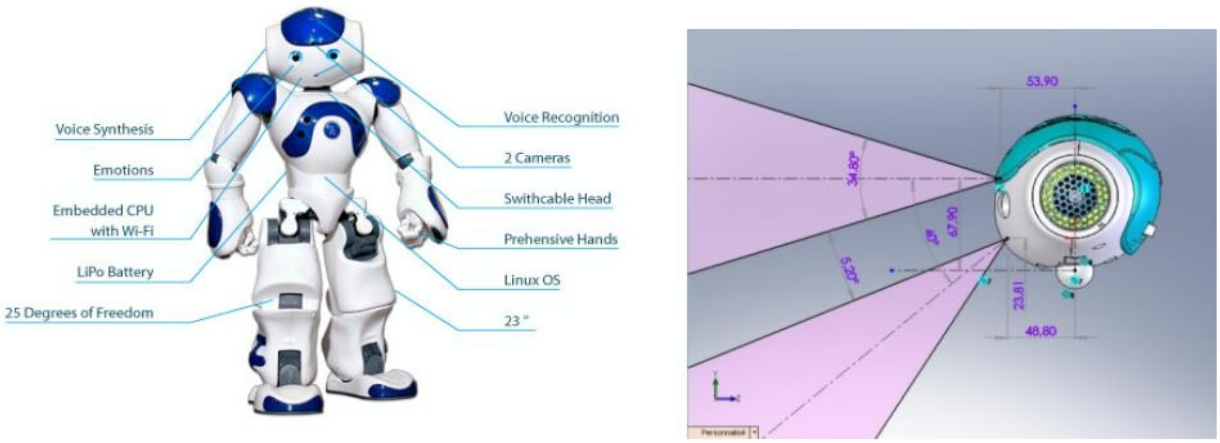


Figura 3.1: Robot Nao de Aldebaran Robotics

3.2. NaoQi

NaoQi es un software que nos provee de un framework para poder desarrollar aplicaciones en C++ y Python. Este framework es ofrecido por Aldebaran Robotics [1].

NaoQi nos permite crear diferentes programas, los cuales están encapsulados en forma de módulos. La funcionalidad del robot está encapsulada en estos módulos, así que pueden comunicarse con los módulos específicos para poder así acceder a los sensores y actuadores.

Todos estos ejecutables, se llaman *broker*, se ejecutan de forma independiente y se conectan a una IP y un puerto. Todos los *broker* pueden ser ejecutados en el robot (usando un compilador cruzado) o desde el ordenador. Algunos de estos *broker* se ejecutan desde el ordenador, ya que, tienen un alto coste de procesamiento, siendo mejor y más rápido ejecutarlos en el ordenador que en el robot.

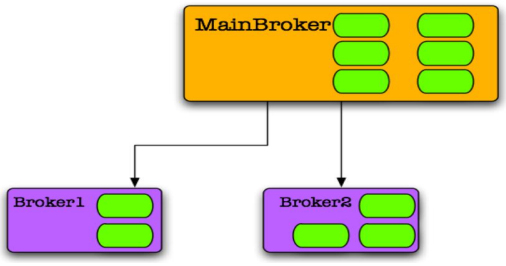


Figura 3.2: Árbol de Brokers

La funcionalidad del *broker* está basada en módulos. Cada *broker* puede tener uno o

[1] http://www.aldebaran-robotics.com

varios módulos. En realidad, los *broker* sólo ofrecen algunos de sus servicios para poder realizar su tarea. Los *broker* se encargan de entregar los mensajes entre los módulos. Además, propocionan la forma de resolver los nombres de los módulos para así evitar especificar la dirección y el puerto del módulo al que se quieren conectar.

Un conjunto de *brokers* se estructuran en forma de árbol. El más importante se llama *MainBroker*. Éste contiene los módulos que dan acceso a los sensores y actuadores del robot, además de otros módulos importantes.

Todos los módulos tienen un API con la funcionalidad que ofrecen. Los *brokers* ofrecen la información de sus módulos y de sus API's a través de servicios web. De forma, que si te conectas a un *broker* usando un navegador, podrás ver que módulos contiene y cual es su API.

Cuando se desarrolla una aplicación compuesta por módulos, se puede desarrollar de dos maneras: como una *librería dinámica* o como un *binario*. Si se crea como *librería dinámica*, los módulos que contiene serán cargados en el *MainBroker*. De esta forma la ejecución es mucho más rápida. La parte negativa es que si alguno de los módulos fallase, también lo haría el *MainBroker*, con los problemas que ello conlleva, como que el robot se caiga al suelo. En cambio, si se crea como un *binario*, si fallase el módulo sólo éste se vería afectado.

Algunos de los módulos que hemos utilizado en este proyecto son: *ALMotion*, *ALTextToSpeech*, *AlAudioPlayer*.

- **ALMotion**. Es el módulo responsable de los actuadores del robot. El API que tiene este módulo nos servirá para mover una parte del robot o todo el robot entero. Los movimientos pueden ser muy simples o muy complicados. Podemos usar llamadas de alto nivel para hacerlo andar, girarse, etc... Si queremos que el robot ande, sólo tenemos que crear un proxy a este módulo, y llamar a una función llamada `walkStraight()`.

- **ALTextToSpeech**. Es el módulo responsable de que el robot pueda hablar. El API que tiene este módulo nos servirá para poder reproducir textos en diferentes idiomas, con diferentes voces, etc...

- **ALAudioPlayer**. Es el módulo responsable de que el robot pueda reproducir ficheros de música.

3.2.1. Coreografo Comercial

En este apartado vamos a comentar el coreografo comercial que nos ofrece Aldebaran Robotics. El coreografo es una aplicación en la cual se puede editar movimientos y comportamientos para el robot Nao. Un comportamiento es un grupo de acciones unidas, basadas en eventos y en una línea de tiempo. La línea de tiempo es la mejor forma de crear y editar estos movimientos.

El coreografo de Aldebaran, mostrado en la figura 3.3, nos ofrece un panel de librerías que están ordenadas por categorías acordes a las funciones que van a desarrollar: movimientos, hablar, reproducción de música, etc.... Estas funciones se pueden arrastar a la línea temporal para crear los comportamientos.

El panel de diagrama de flujo es el lugar donde *componer* los comportamientos para Nao. Para crear un comportamiento, depués de seleccionar la librería que queremos usar tenemos que arrastrarla a la línea de tiempo. La línea de tiempo también nos servirá para poder ordenar los vínculos entre las diferentes librerías y estructurar así un comportamiento.

Exiten varios motivos por los que no se usa este coreografo para realizar la idea de este proyecto. La primera es que está más limitado que el coreografo que se presenta en este proyecto, ya que muchos de los movimientos vienen ya predefinidos y es díficil crear nuevos movimientos desde cero. La segunda razón es que es un coreografo de pago a base de licencias, lo que hace que sea una inversión muy cara. Además, con este proyecto se quiere enriquecer a la arquitectura BICA con esta nueva funcionalidad.

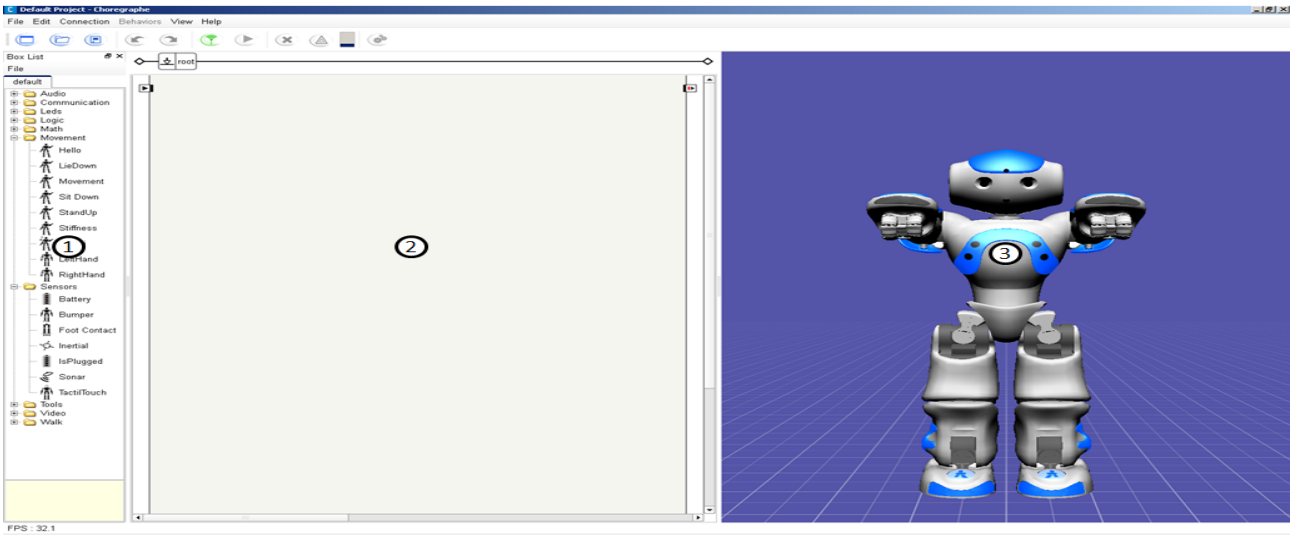


Figura 3.3: Coreografor de Aldebaran Robotics

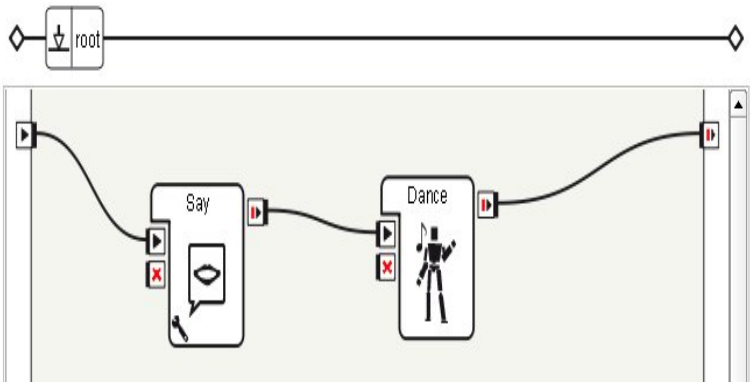


Figura 3.4: Creando un movimiento

3.3. Bica(Behavior Based Architecture For Robot Applications)

Como hemos comentado, NaoQi nos permite desarrollar cualquier funcionalidad con el robot Nao. Siempre que dividamos la funcionalidad que se quiere desarrollar en módulos que se pueden comunicar entre ellos.

Gracias a NaoQi es posible desarrollar comportamientos básicos sólo usándolo, pero no es suficiente para lo que queremos desarrollar. Por ese motivo se ha diseñado una nueva arquitectura en la cual podamos activar y desactivar los componentes.

La unidad básica del funcionamiento de la arquitectura es el *componente*. Cada componente puede ser activado o desactivado. Cuando está activo, el componente estará ejecutando una tarea de manera iterativa a cierta frecuencia y consumiendo recursos. En cambio, cuando está desactivado, el componente no iterará y no consumirá ningún recurso. Un componente puede ser modulado y puede devolver información de la tarea que está ejecutando.

Como se ha comentado cuando un componente esta activo, el componente estará ejecutando una tarea de forma iterativa, esa tarea es la llamada a un método llamado `step()`. Las llamadas a este método hacen que la máquina de estados progrese, ya que cada llamada a `step()` hace que se vuelva a ejecutar su código, sea una máquina de estados o un comportamiento reactivo. Cuando se deja de llamar a `step()`, indirectamente la máquina de estados se para, ya que no se vuelve a ejecutar. La frecuencia con la que se realizan esas llamadas están controladas por otro método llamado `isTime2Run()`. Esta frecuencia puede ser modulada desde otros componentes.

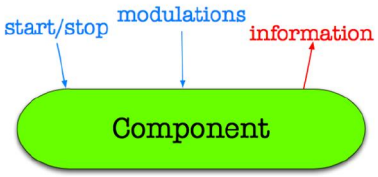


Figura 3.5: Entradas y Salidas de un componente

Un componente, cuando está activo, puede a su vez activar a otros componentes y estos a otros. La activación de los componentes se realiza en forma de árbol. Esta es la idea principal de la arquitectura BICA. Cuando se desarrolla una funcionalidad, ésta se descompone en componentes.

Como se puede ver en la figura 3.6(a), cuando el componente A se activa, éste activa a su vez al componente B y al componente E. El componente B activa al componente C y al componente D. El componente A necesita que el resto de componentes estén activos para realizar tu tarea. En otro ejemplo 3.6(b), el componente A no necesita saber que B ha activado a C y a D. El compomente A sólo está interesado en el componente B y E. Como se puede observar, dos componentes pueden activar a un tercero. Esto es porque ambos pueden querer la información que devuelve.

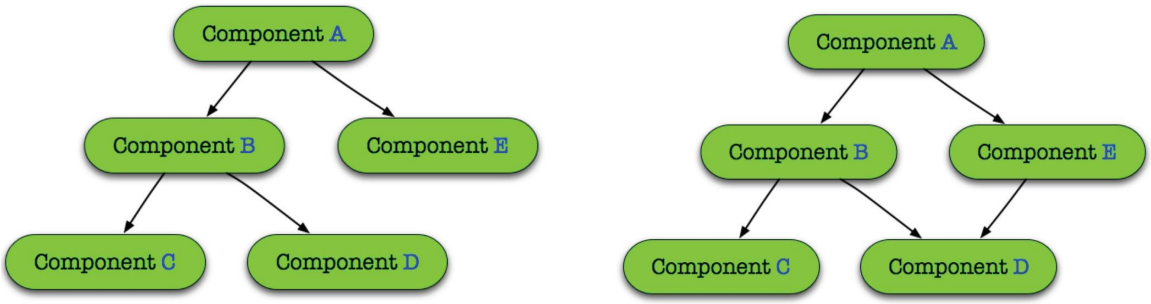


Figura 3.6: Ejemplo 1 y Ejemplo 2

La activación en árbol que hemos visto no es fija cuando se ejecuta en el robot. Los cambios dinámicos dependerán de varios factores: la tarea principal, posición del robot, iteración con humanos, errores...

El robot deberá adaptarse a los cambios de estos factores e ir modulando cada uno de los componentes o activando o desactivando componentes, pasando de una activación de los componentes dinámica a una estática.

Como hemos comentado antes, la idea es descomponer la funcionalidad a desarrollar en varios componentes, los cuales cooperen entre ellos y se puedan comunicar.

Utilizar la arquitectura BICA nos facilita la creación de comportamientos muy complicados. Nos permite desarrollar comportamientos mucho más simples, para que cada uno haga su trabajo y sólo el suyo y se puedan comunicar con el resto de los componentes de una forma fácil. Esta facilidad para conseguir la comunicación entre los comportamientos también nos la ofrece BICA. También nos ofrece una forma de organizar los procesos que se ejecutan en el robot, debido a que cuanto más complejos sean los comportamientos menos secuenciales podrán ser estos.

El comportamiento que se expone en este proyecto, refleja justamente lo que estamos diciendo. Para conseguir el comportamiento final, se han tenido que desarrollar 3 o 4 comportamientos más simples para que realicen sus propias tareas.

En la imagen 3.7 se puede ver como BICA y sus componentes usan NaoQi, y como desde la herramienta JManager, que se explicará en el capítulo 3.4, se comunica con BICA y ésta se comunica con el robot.

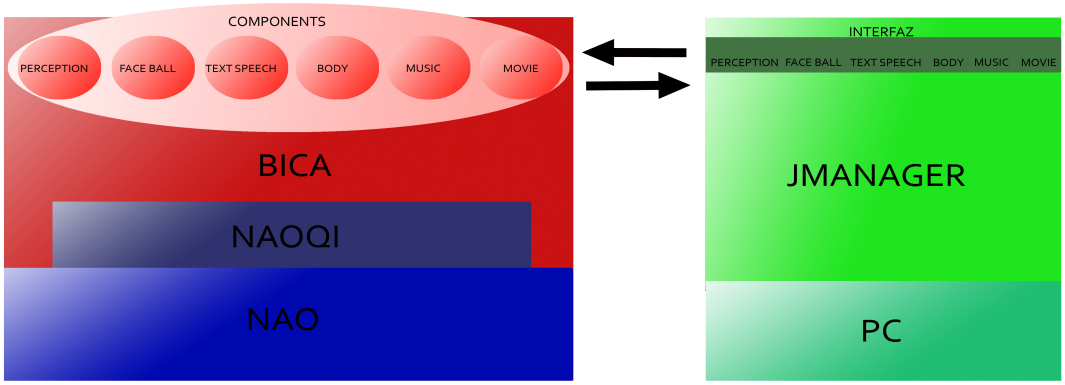


Figura 3.7: BICA, NaoQi, NAO y JManager

3.3.1. VICODE: (VIsual COmponent DEsigner)

El desarrollo de componentes en BICA (figura 3.8) es relativamente sencillo. Sólo es necesario definir una máquina de estados finita y establecer relaciones con otros componentes. VICODE es una herramienta generada en C++ para codificar componentes, tanto básicos como complejos.

En la figura 3.8 se puede ver una máquina de estados finita, con estados definidos y transiciones entre ellos. Se puede observar la conexión entre los componentes, si están

modulados o no, o si requieren información de alguno de ellos.

Esta herramienta genera el código C++ del componente. Esto incluye el código de máquina de estados, la declaración de los componentes utilizados, y las llamadas al método `step()` de los componentes que utiliza o modula. Este método `step()` es el que es llamado iterativamente cuando un componente está activo. Este código se actualiza si desde VICODE se modifica la máquina de estados. Las transiciones entre estados están definidas como funciones que devuelven verdadero o falso según si la transición ha tenido exito o no. Que una transción tenga exito o no, puede depender de otros componentes o de un tiempo estimado.

VICODE está incluida en la herramienta JManager, que hablaremos más adelante. Esta herramienta también nos servirá para activar o desactivar los componentes o poder modularlos.

Como se ha comentado anteriormente, VICODE es la herramienta que se utiliza para realizar los componentes en la arquitectura BICA. Como en nuestro proyecto hemos tenido que desarrollar varios componentes, la herramienta que hemos utilizado para ello ha sido VICODE. Hemos diseñado la máquina de estados de cada uno de los componentes para así agilizar el desarrollo y tener la base que necesita cada componente para estar acorde con la arquitectura BICA y así cumplir uno de los requisitos propuestos.

En el siguiente capítulo podremos ver la máquina de estados de cada uno de los componentes y ver así como se ha dado uso a esta herramienta para el desarrollo del proyecto.

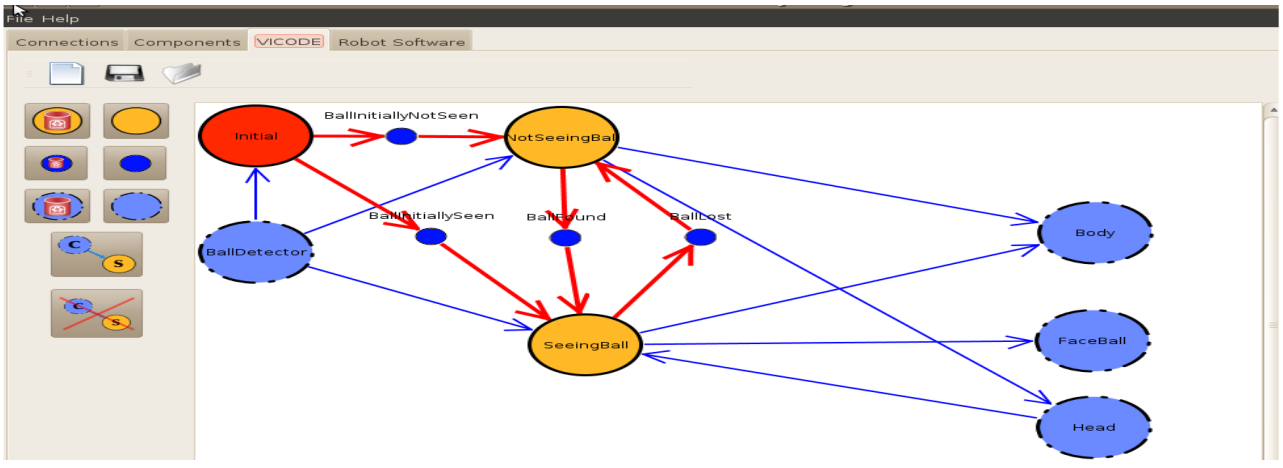


Figura 3.8: Máquina de Estados en VICODE

A continuación podemos ver el ejemplo de dos de los ficheros que hemos generado con

VICODE para la realización de este proyecto.

- **Music.xml**

```
<component id="Music">
<state id="initial" x="4.0" y="4.0" code="Initial_state_code" >
</state>
<state id="Ready" x="230.0" y="7.0" code="Ready_state_code">
</state>
<state id="Playing" x="229.0" y="224.0" code="Playing_state_code">
</state>
<state id="Stop" x="25.0" y="228.0" code="Stop_state_code">
</state>
<transition id="Initial2Ready0" x="163.0" y="51.5" code="Initial2Ready0_transition_code"
source="Initial" target="Ready" priority="1" timer="false"/>
<transition id="Ready2Playing0" x="275.5" y="161.5" code="Ready2Playing0_transition_code"
source="Ready" target="Playing" priority="1" timer="false"/>
<transition id="Playing2Initial0" x="172.5" y="118.0" code="Playing2Initial0_transition_code"
source="Playing" target="Initial" priority="1" timer="false"/>
<transition id="Playing2Stop0" x="163.5" y="337.5" code="Playing2Stop0_transition_code"
source="Playing" target="Stop" priority="2" timer="false"/>
<transition id="Stop2Initial0" x="48.5" y="162.0" code="Stop2Initial0_transition_code"
source="Stop" target="Initial" priority="1" timer="false"/>
</component>
```

- **Music.h**

```
#ifndef Music_H
#define Music_H
#include "Component.h"
#include "Singleton.h"
#include "alptr.h"
#include "alaudioplayerproxy.h"
#include "almotionproxy.h"
#include "albroker.h"
#include <time.h>
class Music : public Component, public Singleton<Music>
{
public:
Music();
~Music();
void init(const string newName, AL::ALPtr<AL::ALBroker> parentBroker);
void step();
```

```
void stop();
private:
static const double twait = 1.5;
static const int Initial = 0;
static const int Ready = 1;
static const int Playing = 2;
static const int Stop = 3;
int state;
void Initial_state_code(void);
void Ready_state_code(void);
void Playing_state_code(void);
void Stop_state_code(void);
bool Initial2Ready0_transition_code(void);
bool Ready2Playing0_transition_code(void);
bool Playing2Initial0_transition_code(void);
bool Stop2Initial0_transition_code(void);
bool Playing2Stop0_transition_code(void);
};
#endif
```

3.4. JManager

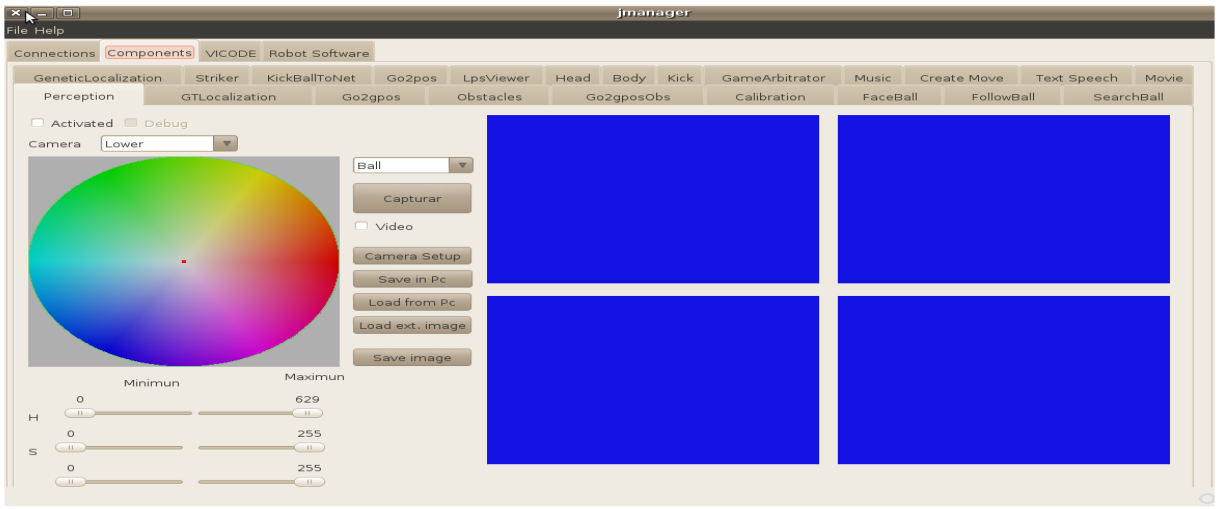
JManager es una aplicación desarrollada en Java. Esta aplicación contiene todas las herramientas para poder realizar tareas de monitorización y configuración en el robot. Esta aplicación se ejecutará en un PC conectandose al robot por cable de red o por la red inalámbrica.

Para poder hacer posible esta comunicación entre el robot y el ordenador, se ha usado un protocolo basado en socket TCP y UDP que se ha desarrollado para BICA. Es un protocolo que está basado en el envío y recepción de mensajes desde JManager a BICA y viceversa.

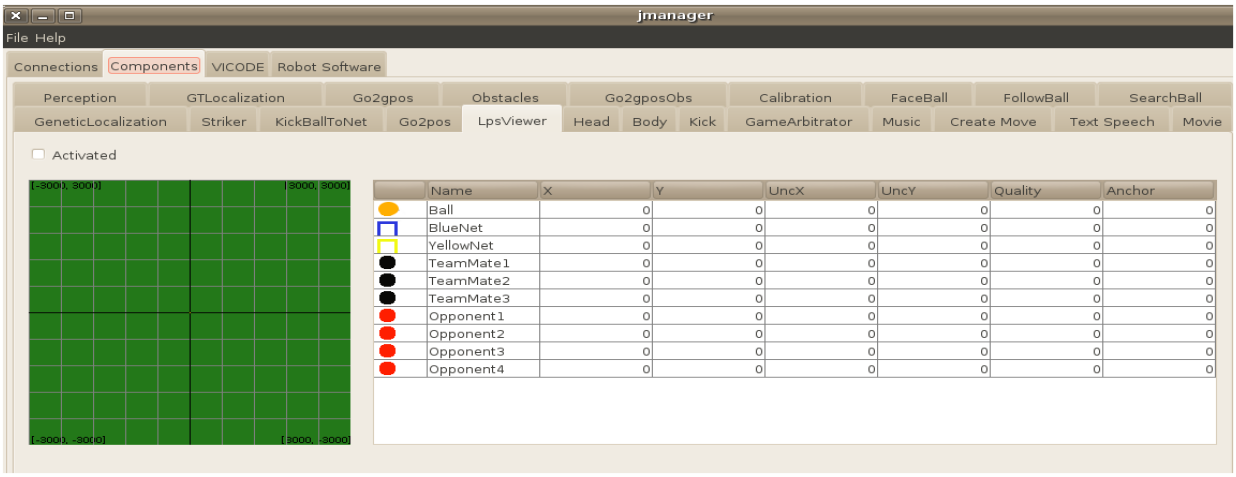
3.4.1. Depuración de los componentes

En JManager podemos depurar los componentes individualmente gracias a que cada uno tiene su pestaña dentro de la herramienta. Podemos activar, modular y cambiar la frecuencia con la que funcionan éstos. Además, podremos obtener información sobre el consumo de la CPU.

En las siguientes figuras, se pueden ver las interfaces gráficas de estas herramientas, en concreto veremos la interfaz del componente *Perception* en la figura 3.9(a) y la interfaz del componente *LpsViewer* en la figura 3.9(b). En ambas, se puede observar un *checkbox* llamado **Activated** que es el encargado de activar o desactivar el componente.



(a)



(b)

Figura 3.9: Componente Perception y Componente LpsViewer

Capítulo 4

Descripción Informática

En este capítulo vamos a describir el método de desarrollo usado para la realización de este proyecto y la implementación del mismo. Describiremos como se ha implementado el comportamiento para la generación de coregrafías basándonos en la plataforma y la arquitectura explicada anteriormente. Se explicará detalladamente el funcionamiento de cada parte del sistema.

Primero describiremos como se ha diseñado el lenguaje para generar las coreografías, viendo que es un lenguaje sencillo e intuitivo para cualquier persona. A continuación describiremos el método de desarrollo, en qué consiste y cómo se ha aplicado a este proyecto. Tras explicar el modelo de desarrollo, nos detendremos a explicar los aportes a la herramienta JManager, para el uso de los componentes. Además, explicaremos la implementación de los componentes que comenzará dando una visión general, para luego pasar a explicar cada una de sus partes.

Por último, veremos la aplicación creada dentro de la herramienta JManager para poder crear movimientos.

4.1. Diseño del lenguaje de guiones

En este apartado vamos a explicar cómo son los guiones que entiende el robot y cómo tienen que ser escritos, que palabras clave utiliza y porqué.

Como hemos comentado antes, gracias a los guiones podemos crear las coreografías que queremos que realice el robot. Las acciones básicas que va a realizar el robot serán moverse, andar, hablar, reproducir música y realizar juegos de luces con los leds.

El idioma que se ha elegido para realizar el *lenguaje* para crear los guiones ha sido el inglés. Esto es así, por la longitud de las palabras que usamos y por el uso a nivel global que se le puede dar a esta aplicación.

A continuación explicaremos los comandos para crear un guión, para después ver un ejemplo completo y explicar que se haría en cada línea.

- **breakpoint**. Cada vez que se introduzca esta instrucción nos servirá para poder volver o avanzar a esta línea mientras se esta ejecutando un guión. Es un comando muy útil, cuando en algún punto de la coreografía queramos volver a repetir un punto, o saltarnos algo que ya no tiene sentido que ocurra o que diga el robot.

- **music**. Mediante este comando ejecutaremos los ficheros de música. Como segundo parametro habrá que indicar la ruta en el robot donde estará el fichero de música que queremos reproducir. La forma de escribirlo en el fichero del guión es:

  `music /HOME/file.mp3`.

- **talk**. Mediante este comando haremos que el robot hable. Después del comando hay que escribir dos parametros más. El primero será *file*, si queremos que lea lo que está escrito en un fichero de texto, o *text*, si lo que queremos es escribir la frase en el guión. Y el último parámetro será, si se ha elegido *file*, la ruta del fichero de texto en el robot. Si se ha elegido *text*, la frase en el idioma deseado. Por otro lado, si el segundo parametro es *language*, *volume* o *voice*, entonces el tercer parametro será el idioma, el valor del volumen o el tipo de voz, respectivamente. La forma de escribirlo en el fichero del guión es:

  ```
  talk text Hola como estas
  talk file /HOME/file.txt
  talk language spanish
  talk volume 55
  talk voice namevoice
  ```

- **walk**. Mediante este comando haremos que el robot ande o gire. Este comando necesita 3 parametros, *velocidad líneal*, *velocidad de rotación* y *velocidad Z*. Para dejar de andar habrá que escribir de nuevo este comando pero con las tres velocidades a 0. La forma de escribirlo en el fichero del guión es:

  ```
  walk 0.7 0.1 0.3
  walk 0 0 0
  ```

- **mov**. Mediante este comando ejecutaremos los movimientos o bailes en el robot. Necesita sólo un parámetro y es el nombre del movimiento, que ya ha tenido que se cargado en el robot previamente. La forma de escribirlo en el fichero del guión es:

  `mov muevebrazos`

- **led**. Mediante este comando podremos cambiar los colores de los leds del robot. Este comando necesita 3 parámetros. El primero es el identificador del led, que se puede observar en la ayuda. El segundo parámetro definirá el modo en el que se pondrá ese led, que podrá ser apagado, encencido o automático. Y por último, se escogerá el valor del color que también se podrá ver en la ayuda. La forma de escribirlo en el fichero del guión es:

  `led 5 1 1` Este ejemplo, pondrá los ojos de color rojo.

  `led 0 2` Este ejemplo, pondrá todos los leds de colores aleatorios.

  `led 0 0` Este ejemplo, apagará todos los leds.

- **wait**. Mediante este comando podremos esperar a que terminen algunas de las tareas que se están realizando, esperar un tiempo determinado o esperar a que se pulse uno de los 3 botones del robot. Este comando necesita 2 parámetros. El primero será *task*, si vamos a esperar a que termine una tarea, *time*, si lo que vamos a esperar es un tiempo, definido en segundos o *press*, si vamos a esperar a que alguien pulse uno de los 3 botones. Si lo que queremos es esperar a que acabe una tarea, como hemos dicho, hay que usar el comando *task* y el tercer parámetro será el nombre de esa tarea, music, text, mov... Si lo que queremos es esperar un tiempo determinado, el tercer parámetro será el tiempo en segundos. Por último, si lo que queremos es esperar a pulsar uno de los 3 botones, el tercer parámetro será *left* (pie izquierdo), *right* (pie derecho) o *chest* (pecho). La forma de escribirlo en el fichero del guión es:

  `wait task mov` Esperará a que termine el movimiento que se está realizando.

  `wait time 10` Esperará a que pasen 10 segundos.

  `wait press left` Esperará a que alguien pulse el botón del pie izquierdo del robot.

A continuación vamos a ver un ejemplo de un guión completo, donde podremos ver cada una de los comandos:

```
breakpoint
talk language spanish
```

```
talk text Hola soy Nao y esto es una Demo
mov muevebrazos
music /home/nao/mp3/cancion.mp3
wait task mov
breakpoint
mov baila
leds 1 1 2
leds 0 2
wait task mov
stop task music
breakpoint
talk file /home/nao/text/texto.txt
walk 0.5 0.5 0.5
talk text Estoy andando mientras hablo contigo
wait time 4
walk 0 0 0
breakpoint
mov sentarse
wait press left
stiffness off
```

Es importante destacar que aunque el guión se debe escribir de forma secuencial, la ejecución de los comandos no lo es. Las tareas se ejecutaran de forma paralela, hasta que aparezca el comando `wait`, entonces la ejecución esperará a que termine la tarea o pase el tiempo correspondiente.

A continuación explicaremos cada línea.

1. `breakpoint` - Marcará un punto de ruptura para poder volver a él cuando se desee.
2. `talk language spanish` - Definimos el idioma en el que queremos que hable el robot.
3. `talk text Hola soy Nao y esto es una Demo` - El robot dira: 'Hola soy Nao y esto es una Demo'.
4. `mov muevebrazos` - Realizará el movimiento llamado 'muevebrazos'.
5. `music /home/nao/mp3/cancion.mp3` - Reproducirá el fichero cancion.mp3 que estará en la ruta /home/nao/mp3 dentro del robot.
6. `wait task mov` - Esperará hasta que termine de realizar la tarea `mov` que es el movimiento llamado 'muevebrazos'.

7. `breakpoint` - Marcará un punto de ruptura para poder volver a él cuando se desee.
8. `mov baila` - Realizará el movimiento llamado 'baila'.
9. `leds 5 1 2` - Activará los leds de los pies con el color azul.
10. `leds 0 2` - Activará todos los leds de forma automática y con colores aleatorios.
11. `wait task mov` - Esperará hasta que termine de realizar la tarea `mov` que es el movimiento llamado 'baila'.
12. `stop task music` - Detendrá la reproducción del fichero cancion.mp3.
13. `breakpoint` - Marcará un punto de ruptura para poder volver a él cuando se desee.
14. `talk file /home/nao/text/texto.txt` - - El robot dirá las frases que estén escritas en el fichero text.txt dentro de la ruta /home/nao/text/ dentro del robot.
15. `walk 0.5 0.5 0.5` - El robot empezará a andar con una velocidad *Vx=0.5 Vy=0.5 Vz=0.5*.
16. `talk text Estoy andando mientras hablo contigo` - El robot dira: 'Estoy andando mientras hablo contigo'.
17. `wait time 4` - Esperará hasta que pasen 4 segundos.
18. `walk 0 0 0` - El robot dejará de andar.
19. `breakpoint` - Marcará un punto de ruptura para poder volver a él cuando se desee.
20. `mov sentarse` - Realizará el movimiento llamado 'sentarse'.
21. `wait press left` - Esperará hasta que alguien pulse el botón del pie izquierdo del robot.
22. `stiffness off` - Desactiva la fuerza de los motores del robot.

4.2. Desarrollo de los Componentes y de la interfaz dentro de JManager

En este apartado vamos a explicar la ampliación de la herramienta JManager que nos servirá para usar los componentes desarrollados en este proyecto. De cada componente que

ha sido necesario desarrollar, podremos entender como es su funcionamiento, ver como es su máquina de estados finita, y que hace en cada estado y como se usa la herramienta JManager.

4.2.1. Componentes

En este apartado vamos a explicar el funcionamiento de los componentes que se han desarrollado para este proyecto. Este es el punto central del proyecto, junto con el diseño del lenguaje, donde se encuentran las mayores aportaciones. Aunque en el capítulo anterior hemos explicado JManager, en este apartado seguiremos hablando de él ya que lo usaremos para ver como se activan, desactivan o se configuran cada uno de los componentes.

Cuando se planifica un componente normalmente se va planificar también un cambio en la interfaz de JManager para poder realizar las pruebas y poder activar o desactivarlo o para poder usarlo como un componente aislado.

El componente principal de este proyecto, se llama `Movie`, vease la figura 4.1. Y se llamará así, porque es el que se encarga de interpretar los guiones y de reproducirlos en el robot Nao.

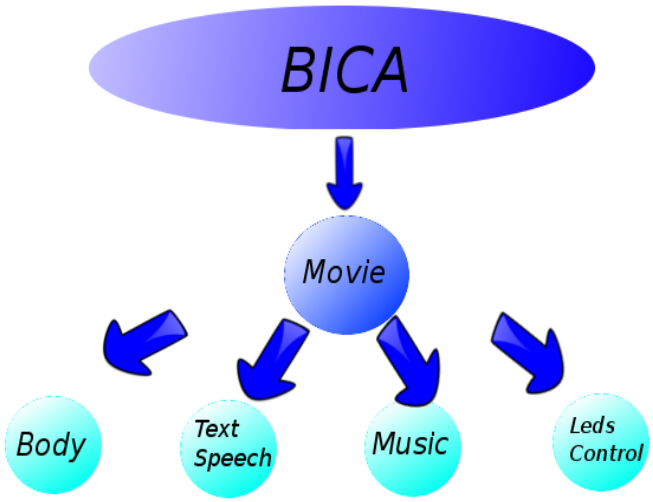


Figura 4.1: Compente Movie dentro de BICA

Este componente recogerá información y modulará otros componentes más específicos. Y juntos, serán capaces de realizar los movimientos, reproducir la música, hablar, etc...

Los componentes más específicos de los que estamos hablando son:

- `Body`. Se encargará de realizar los movimientos en el robot.
- `TextSpeech`. Se encargará de reproducir de forma hablada los textos.
- `Music`. Se encargará de reproducir los ficheros de música.
- `LedsControl`. Se encargará de la iluminación de los leds del robot.

Cada uno de estos componentes son máquinas de estados y más adelante veremos su funcionamiento individual y como son modulados y como ofrecen información al componente principal, `Movie`.

Music

Este componente es uno de los componentes que será modulado por el componente `Movie`. También, se podrá utilizar en solitario desde la herramienta JManager. La idea principal de este componente es la reproducción de ficheros de música en formato MP3 o WAV. Desde la interfaz se podrá elegir el fichero de música que se quiera reproducir en el robot.

Este componente está formado por los siguientes estados, mostrados en la figura 4.2:

- Initial
- Ready
- Playing
- Stop

El estado *Initial*, es el estado desde el cual parte la máquina de estados cuando se activa el componente. La máquina de estados no avanzará de estado hasta que desde JManager o desde otro componente se module. Para modular este componente hay que seleccionar un fichero de música desde JManager o indicarle la ruta del fichero al componente. En este caso, será el componete `Movie` el que le diga el fichero de música que tiene que reproducir. Este fichero será el que se haya escrito en el guión.

Como se ha dicho, para poder avanzar en la máquina de estados del estado *Initial* al estado *Ready* se debe modular el componente. Para ello, el componente `Music` tiene una función llamada `setFileMP3()` a la cual se le dirá cual es la ruta del fichero de música que hay que reproducir.

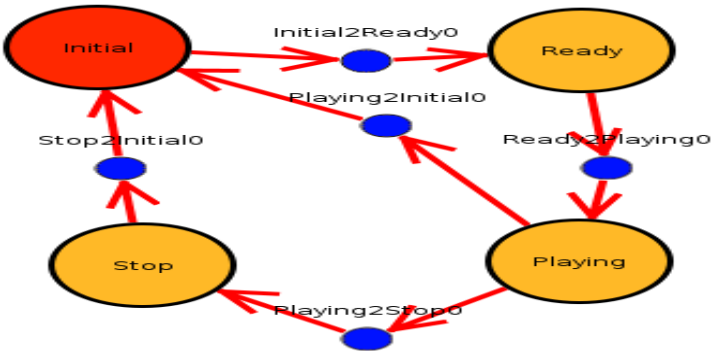


Figura 4.2: Máquina de Estados Music

Cuando el fichero ha sido seleccionado la máquina de estados avanzará al estado *Ready* y ya estará preparado para poder empezar a reproducir la música. Para avanzar en la máquina de estados, el componente deberá volver a ser modulado, bien desde JManager, pulsando el botón *Play* de la interfaz o bien modulado por otro componente. Al igual que en antes, será el componente `Movie` quien module de nuevo.

Para modular el componente desde `Movie` se llamará a una función llamada `playPress()`. Esta función hará que el compomente `Music` cree una conexión con el robot Nao y, esta vez sí, le indique qué fichero de música tiene que reproducir. Para ello, llamará a la función `musicmp3->pCall("playFile",fileMp3)`. Esta función nos devolverá un identificador con el cual podremos parar la música cuando lo deseemos.

Por último, la música se parará cuando se module el componente o desde la interfaz se pulse el botón *Stop*. Esto hará que de nuevo se use la conexión con el proxy al robot, pero esta vez será para detener la música. La función a la que se llama es `musicmp3->callVoid("stop",IdSong)`, donde `IdSong`, es el identificador del que hablabamos antes.

El componente `Movie` se nutrirá de la información de este componente a través de una función que le dirá si ha terminado o no de reproducir el fichero de música.

En la siguiente imagen 4.3 podemos ver el interfaz para este componente en JManager que se usa para monitorizarlo y depurarlo.

- **Activate**. Este checkbox nos servirá para activar o desactivar el componente. De manera que, aunque se pulse el botón Play, si el componente no está activo no se hará nada.
- **Load Audio MP3** . Este botón, como el nombre indica, nos servirá para cargar un

fichero en la interfaz y modular el componente `Music`. Es decir, le dirá la ruta del fichero en el robot a la función `setFileMP3()`.

- **Play**. Pulsando este botón modularemos el componente para, una vez que se ha elegido el fichero que se quiere reproducir, el robot empiece a reproducirlo.
- **Stop**. Si el botón Play ha sido pulsado, modulará el robot para parar la música y dejar de reproducirla.

Figura 4.3: Componente Music

TextSpeech

`TextSpeech` es otro de los componentes que será modulado por el componente `Movie`. También se podrá utilizar en solitario desde la herramienta JManager. La idea principal de este componente es la reproducción mediante el *habla* de frases a través del robot. Desde la interfaz se podrá elegir el volumen, el idioma en el que queremos que el robot nos hable, el tipo de voz, etc...

Este componente está formado por los siguientes estados, mostrados en la figura 4.4:

- Initial
- Speech

El estado *Initial* es el estado desde el cual parte la máquina de estados cuando se activa el componente. La máquina de estados no avanzará de estado hasta que desde JManager o

desde otro componente se module. Para modular este componente hay que enviar la frase en el idioma que se haya seleccionado previamente. En este caso, será el componente `Movie` el que le module con la frase tiene que reproducir.

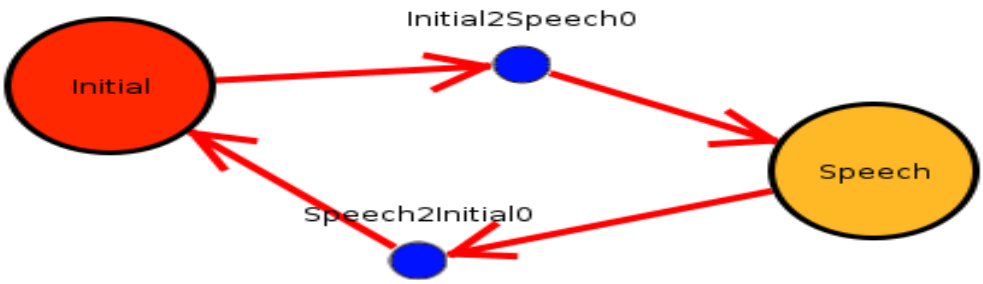


Figura 4.4: Máquina de Estados TextSpeech

Como se ha dicho, para poder avanzar en la máquina de estados del estado *Initial* al estado *Speech* se debe modular el componente. Para ello, el componente `TextSpeech`, tiene una función llamada `setmiText()`, a la cual, se le dirá cuál es la frase que hay que reproducir. Cuando ocurre esto, el robot reproduce la frase según el idioma. Una vez termina de reproducir vuelve al estado *Initial* esperando a volver ser modulado.

Este componente, además de poder modular lo que queremos que diga el robot, tiene otras formas de modulación, como son:

- **Volumen**. Podremos modular el volumen al que queremos que el robot hable. Eso se hará usando la función `setVolume()`, pasándole el volumen con un valor entre 0-1.
- **Voz**. Podremos modular el tipo de voz con la cual el robot habla. Para ello habrá que usar la función `setVoice()`, pasándole el nombre de la nueva voz.
- **Efecto de la Voz**. Podremos modular el tono de la voz, de más a menos aguda.
- **Idioma**. Por último, podremos modular el idioma del robot. Para ello usaremos la función `setLanguage()`, indicándole el idioma con el cual queremos que el robot hable.

El componente Movie se nutrirá de la información de este componente a través de una función que le dirá si ha terminado o no de hablar.

En la figura 4.5 podemos ver el interfaz para este componente en JManager que se usa para monitorizarlo y depurarlo.

- **Activate**. Este checkbox nos servirá para activar o desactivar el componente. De manera que, aunque se pulse el botón SendText, si el componente no está activo, no se hará nada.
- **Get Value**. Este botón nos servirá para obtener los valores que están actualmente en el robot y saber el volumen, el idioma y el tipo de voz.
- **Volume**. Con este *slider* podremos modificar el volumen en el que habla el robot.

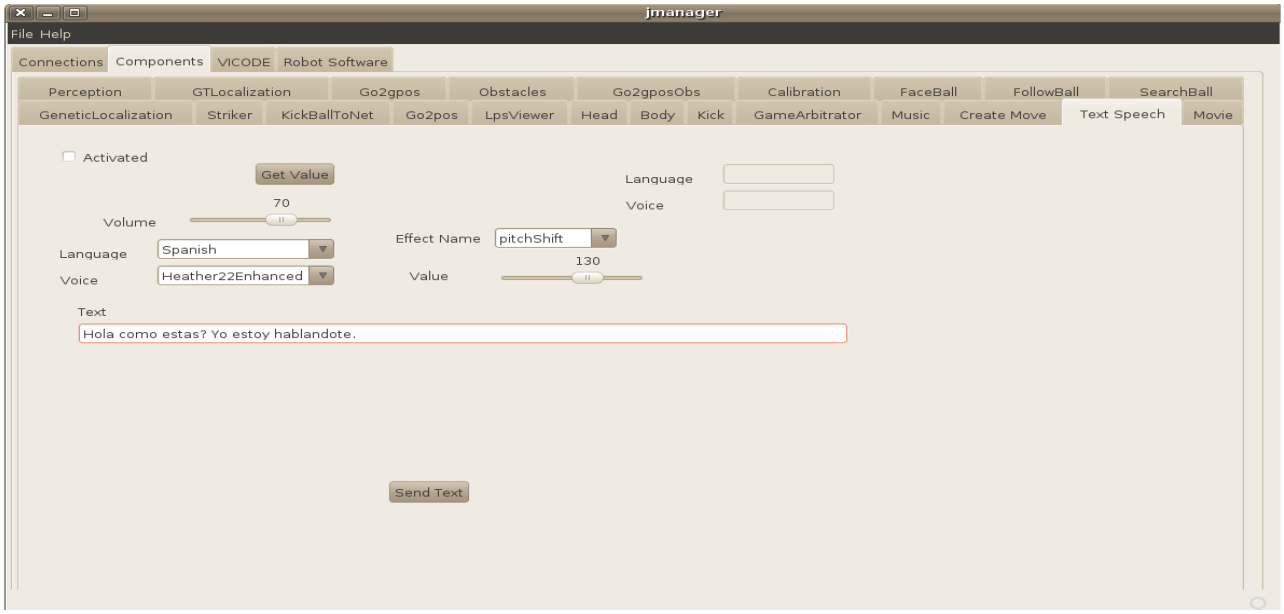


Figura 4.5: Componente TextSpeech

- **Languaje**. Mediante este desplegable podremos elegir el idioma en el cual queremos que el robot hable.
- **Voice**. Con este desplegable podremos elegir la voz con la cual queremos que hable el robot.
- **Effect Name** y **Value**. Con la combinación de estos dos valores podremos modificar los valores agudos del robot para tener un tono de voz diferente.
- **Text**. Es el campo donde vamos a introducir la frase que queremos que el robot nos diga.
- **Send Text**. Mediante este botón enviaremos el texto que este en *Text*, siempre y cuando el componente este activo.

Body

Este componente ya estaba diseñado por parte del grupo de Robótica y debido a que es el que realiza los movimientos del robot, se ha reutilizado para este proyecto.

`Body` es uno de lo componentes más importantes. También será modulado por `Movie`. Al igual que el resto de componentes, éste también se podrá utilizar en solitario desde la herramienta JManager. La idea principal de este componente es la realización de movimientos en el robot. Dentro de los movimientos que pueda realizar el robot tendremos dos aspectos, *andar* y *resto de movimiento*.

Este componente está formado por los siguientes estados, mostrados en la figura 4.6:

- Initial
- Stopped
- Walking
- Moving

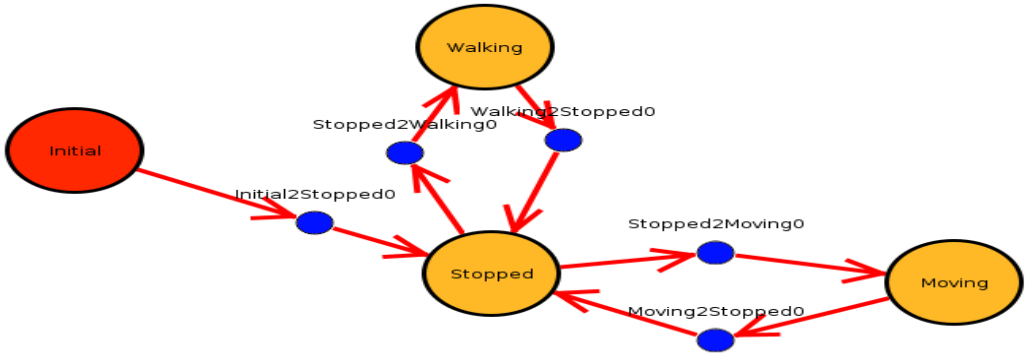


Figura 4.6: Máquina de Estados Body

El estado *Initial* es el estado desde el cual parte la máquina de estados cuando se activa el componente. La máquina de estados no avanzará de estado hasta que desde JManager o desde otro componente se module. Este estado en este componente es un estado peculiar ya que no espera ni hace nada. Cuando arranca avanza directamente al estado *Stopped*.

Una vez en el estado *Stopped*, el componente tendrá que ser modulado para poder avanzar de estado. En este caso, podrá avanzar a dos estados diferentes, dependiendo del tipo de modulación que reciba. Los estados a los que puede avanzar son: *Walking* o *Moving*.

Para avanzar al estado *Walking* hay que modular el componente mediante dos parámetros: *velocidad lineal(v)* y *velocidad de rotación(w)*. Ambos parámetros aceptan valores entre [-1,1]. Si v es 1, el robot anda de forma recta; si v es -1, el robot anda hacía detras y si v es 0, el robot no andará. Si w es 1, el robot girará hacía la izquierda; si w es -1, el robot girará hacía la derecha y si es 0, el robot no girará.

Para volver al estado *Stopped* tan sólo hay que modular el componente con las velocidades a 0.

Para avanzar al estado *Moving* hay que modular el componente mediante un sólo parametro: *nombre del movimiento*. Cuando arrancamos el programa en el robot, uno de los primeros pasos es cargar los movimientos predefinidos en el robot para que cuando los queramos ejecutar la carga de éstos en el robot sean más rapida.

Más adelante, hablaremos de la forma de crear movimientos que se ha desarrollado también en este proyecto mediante una de las aplicaciones de la herramienta JManager.

Cuando se modula el componente para realizar un movimiento, como hemos dicho, se carga en el robot y éste lo ejecuta.

El componente `Movie` se nutrirá de la información de este componente a través de una función que le dirá si ha terminado o no, de realizar un movimiento.

En la siguiente imagen 4.7 podemos ver el interfaz para este componente en JManager que se usa para monitorizarlo y depurarlo.

- **Activate**. Este checkbox nos servirá para activar o desactivar el componente.
- **Walk**, **Backward**, **RotateLeft**, **RotateRigth**, **Stop**. Mediante estos botones podremos hacer que el robot avance, retroceda, rote a la izquierda, rote a la derecha o se pare.
- **PoseInit**. Este botón nos servirá para colocar al robot en la posición inicial, que será una posición estable para él.
- **Debug Kick**. Este checkbox nos servirá para activar o desactivar las funciones de depuración de movimientos dentro del componente.
- **Left Kick**. Este botón nos servirá para realizar un chut con la pierna izquierda. El chut será paramétrico. Esto significa que el chut lo realizará especificando la posición de una articulación en coordenadas cartesianas con respecto a un eje de referencia.

- **Right Kick**. Este botón nos servirá para realizar un chut con la pierna derecha. El chut también será paramétrico.
- **doMove**. Este botón nos servirá para ejecutar un movimiento. Le diremos que movimiento debe ejecutar escribiendo el nombre del movimiento en la casilla de texto. El movimiento ha tenido que ser cargado antes en el robot.

Figura 4.7: Componente Body

Led Control

Por último, hablaremos de otro de los componentes que también será modulado por el componente `Movie`. Este componente a diferencia del resto, sólo se podrá modular desde otro componente. Por lo tanto, no tendrá interfaz en JManager. La idea principal de este componente es el uso de los leds del robot para poder componer juegos de luces con ellos.

Este componente está formado por los siguientes estados, mostrados en la figura 4.8:

- Initial
- SetUpdate
- Change

El estado *Initial* de este componente se comporta igual que el componente `Body`. Es un estado por el que se pasa sólo cuando se activa el componente. Una vez se ha activado, avanza inmediatamente al estado *SetUpdate*. En este estado estará hasta que se modifique el valor de alguno de sus leds.

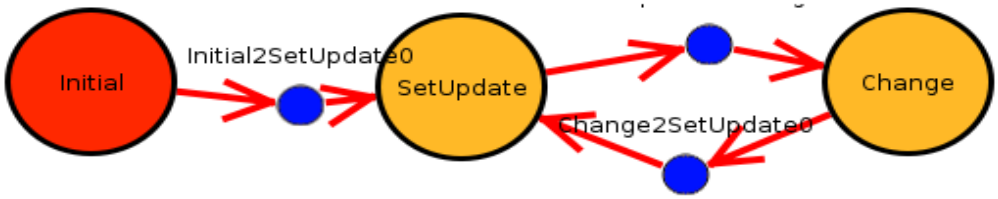


Figura 4.8: Máquina de Estados Led Control

Para modular este componente se necesitan tres parametros: el identificador del *led*, la *activación* y el *color*. Para poder elegir el led que queremos modular podremos usar la ayuda que tenemos en el interfaz del componente `Movie`, será un identificador único para cada led. El parámetro de activación será un valor entre [0-2], siendo 0 apagado, 1 encencido y 2 automático. Por último, tendremos el valor del color en el que se encendará el led. Este valor se podrá omitir cuando el valor de la activación sea 0 o 2.

Cuando se activan los leds avanzamos al estado *Change*. En este estado se hace una actualización de los valores de los leds y se vuelve al estado *SetUpdate* a esperar de nuevo que sea modulado.

Desde este componente `Movie` no se obtendrá información de cuando ha terminado o no, ya que, la duración de las animaciones de los leds está definido por un tiempo finito.

Movie

La idea principal de este componente es la reproducción del guión. Este componente se encargará de modular los componentes necesarios para poder realizar movimientos, hablar, reproducir música, etc... Desde la interfaz del componente se podrán cargar o crear los guiones e interactuar con ellos.

Como se ha explicado en un apartado anterior, los componentes son máquinas de estados finitas. Este componente está formado por los siguientes estados, mostrados en la figura 4.9:

- Initial
- Set File
- Running
- Waiting

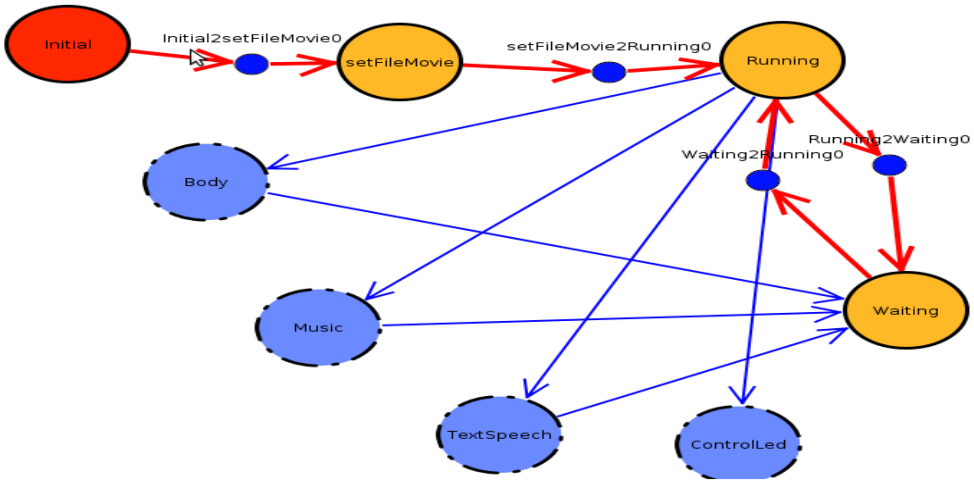


Figura 4.9: Máquina de Estados Movie

El estado *Initial* es el estado desde el cual parte la máquina de estados cuando se activa el componente. La máquina de estados no avanzará de estado hasta que desde JManager o desde otro componente se module. Para modular este componente hay que seleccionar un fichero de texto desde JManager, en el cual estará el guión escrito o indicarle la ruta del fichero al componente.

Como se ha dicho, para poder avanzar en la máquina de estados del estado *Initial* al estado *Set File* se debe modular el componente. Para ello, el componente `Movie` tiene una función llamada `setFile()`, a la cual se le dirá cual es la ruta del fichero que hay que interpretar. Esta función, además se encargará de preparar la estructura necesaria para encontrar los posibles *Breakpoint* que pueda tener el guión.

Una vez que se ha seleccionado el fichero, la máquina de estados avanzará al estado *Set File*. Al igual que antes, la máquina de estados permanecerá en este estado hasta que vuelva a ser modulada, bien desde JManager pulsando el botón *Play* de la interfaz y eligiendo la línea donde se desea comenzar (esto no es obligatorio, por defecto empezará en la línea 1), como se puede ver en la imagen 4.10, o bien sea modulado por otro componente.

Para modular el componente estando en el estado *Set File* se llamará a una función llamada `pressPlay()`. A esta función se le pasará la línea desde la cual se quiere empezar a ejecutar el guión. Cuando no sea la primera línea, se leerá el fichero hasta llegar a esa línea y se podrá cambiar el estado al siguiente, *Running*.

El estado *Running* es el estado que se encarga de procesar el fichero y de identificar que comandos son los que hay que ejecutar. En este estado se estará todo el tiempo leyendo del guión y obteniendo los comandos. Estará leyendo sin parar hasta que, o bien se termine el

guión y acabe la coreografía, o bien encuentre un comando llamado *wait*. Esto hará que la máquina de estados avance al estado *Waiting*.

Como hemos comentado, en el estado *Running* se estará leyendo el fichero del guión. Cada línea que se lee es una instrucción que tiene que realizar el robot. Para obtener esta instrucción hay que procesar el guión. Esto se hace a través de la función `getCommand`. Desde esta función se modularán el resto de componentes que se utilizan para realizar las coreografías.

Como hemos comentado, cuando estamos en el estado *Running* y en la función `getCommand()` nos encontramos con la instrucción `wait`, pasamos al estado *Waiting*. En este estado el componente estará esperando hasta que algunas de las tareas o bien un tiempo determinado acabe. De esta forma, podremos esperar a que el robot termine de realizar un baile o termine de hablar antes de realizar otro movimiento o empezar a decir otra frase diferente.

Sólo volveremos del estado *Waiting* al estado *Running* cuando sepamos que la tarea o el tiempo que estamos esperando ha termiando. Si se trata de tiempo por lo que estamos esperando, en la función que se encarga de controlar la transición entre el estado *Waiting* y el estado *Running* se controlará el tiempo desde que se inicio la espera hasta que pasen los segundos especificados. Si la espera es por una tarea será el componente encargado de realizar esta tarea quien nos dirá si ha terminado de realizarla o no. En este punto, es donde se puede comprobar como se utiliza la información que devuelven los componentes.

Las tareas por las que se pueden esperar a que las termine el robot serán: que termine de hablar, termine de realizar un movimiento o termine de reproducir un fichero de música. Otra de las tareas por las que se puede esperar no depende sólo del robot. También puede depender de la interacción con una persona. El componente podrá estar esperando a que se pulse uno de los 3 botones que tiene el robot, 2 en el pies y 1 en en pecho, de manera que hasta que no se pulse el botón que se ha definido en el guión seguiremos en el estado *Waiting*. Esto es adecuado, para casos en los que no se sepa el tiempo que vamos a querer que el robot esté esperando.

En la figura 4.10 podemos ver el interfaz para este componente en JManager que se usa para monitorizarlo y depurarlo.

- **Activate**. Este checkbox nos servirá para activar o desactivar el componente. De manera que, aunque se pulse el botón Play si el componente no está activo no se hará nada.

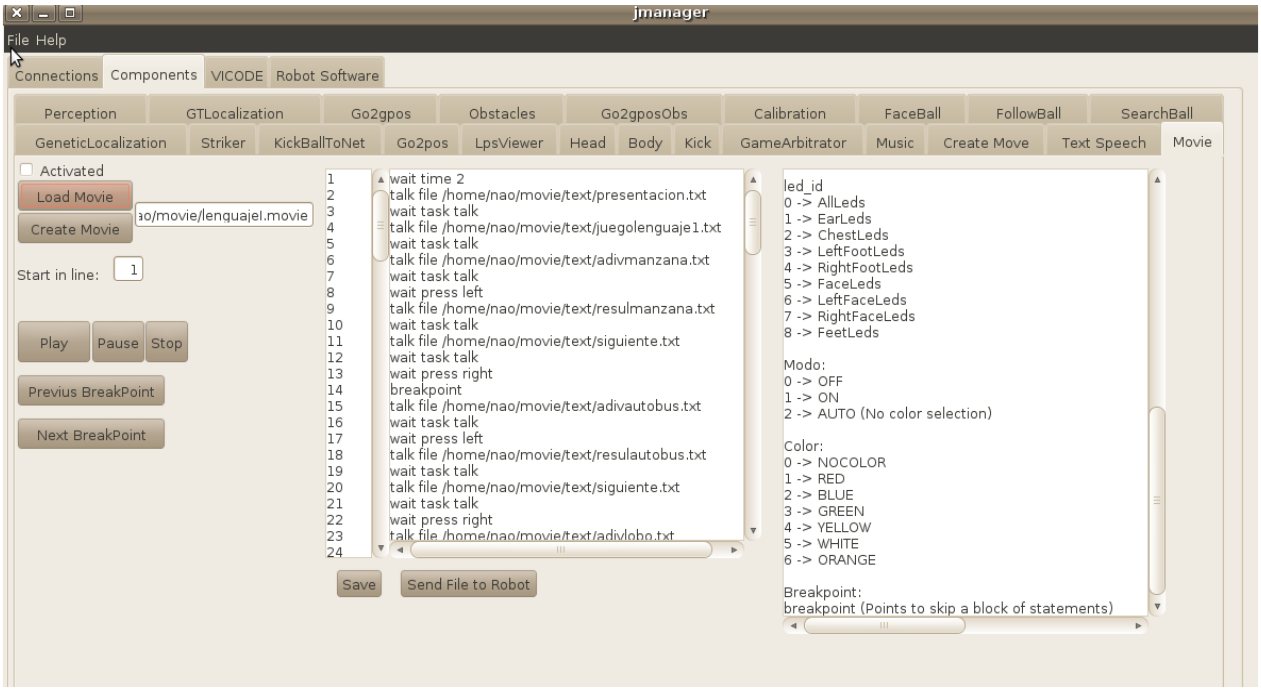


Figura 4.10: Componente Movie

- **Load File**. Este botón como su nombre indica, nos servirá para cargar un fichero en la interfaz y modular el componente `Movie`. Es decir, le dirá la ruta del fichero en el robot a la función `setFile()`.

- **Create File**. Este botón nos servirá crear un nuevo fichero y poder escribir en el interfaz un nuevo guión. Cuando se crea un guión hay que enviarlo al robot usando el botón **Send To Robot**.

- **Start Line**. Será la línea por la que empiece a reproducirse el guión.

- **Play**. Como se ha comentado antes, este botón se encargará de modular el componente `Movie` haciendo que si el fichero ha sido cargado correctamente se pueda empezar a ejecutar la coreografía.

- **Pause**. Este botón pondrá en pause el comportamiento y obtendrá la línea por la cual iba en ese momento para cuando se vuelva a pulsar Play, empiece por esa línaa.

- **Stop**. Este botón hará que se detenga la coreografía, exceptuando los movimientos que este realizando el robot. Esto es así para evitar caidas de éste ya que, cuando se pulse, puede estar en una posición peligrosa y caer al suelo. En cambio, la reproducción de ficheros de música o el habla se detendrán.

- **Previous BreakPoint**. Este botón nos servirá para movernos hacia atrás en el guión según los *breakpoint* que se hayan incluido en el guión. Cuando se pulse se parará el compomente y esperará a que pulsemos de nuevo Play y arrancará en la línea donde estuviese el *breakpoint*.

- **Next BreakPoint**. Al igual que el botón anterior nos servirá para movernos hacía delante dentro del guión. Hará la misma operación que el botón anterior.

- **Save**. Este botón nos servirá para guardar el guión en la ruta que hayamos elegido. Sólo guarda la información en la ruta del PC.

- **Send To Robot**. Este botón envía al robot el fichero y lo guarda en la ruta que hayamos definido, además de guardalo en el PC.

- **Editor de guiones**. En este apartado podemos editar un guión que ha sido cargado o crear un guión nuevo. Después de modificarlo podemos como hemos visto o guardarlo simplemente o enviarselo al robot.

- **Ayuda**. En la parte izquierda, como podemos ver en la figura 4.10, tenemos la ayuda del lenguaje que se ha creado para generar los guiones.

En la figura 4.11 se puede observar como es la interacción entre JManager (desde el interfaz del componente `Movie`) y el robot.

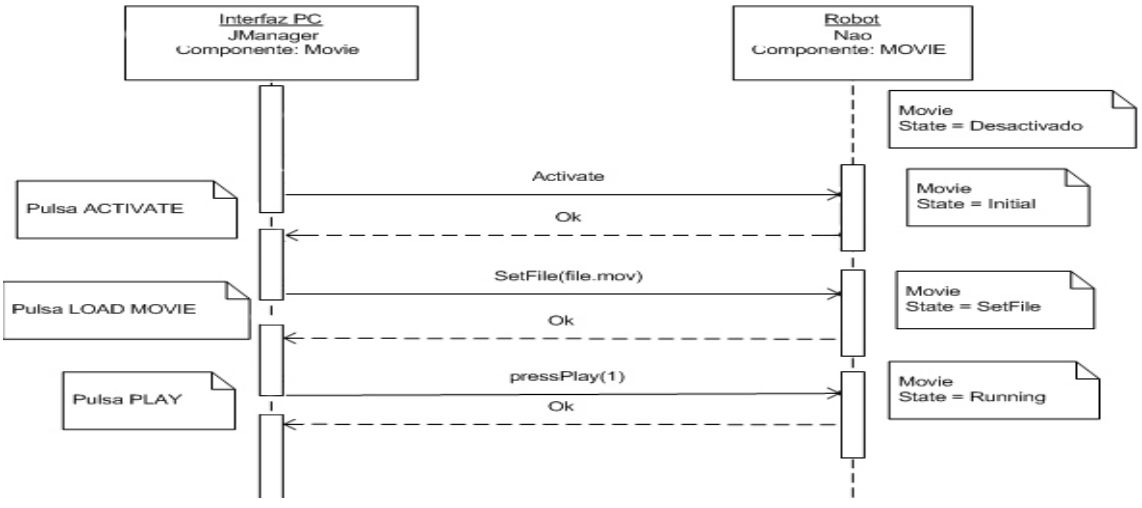


Figura 4.11: Interacción entre JManager y el Robot

4.2.2. Herramienta para crear movimientos

En este apartado vamos hablar de la herramienta que hemos desarrollado para poder generar movimientos para el robot.

Como se ha comentado en un apartado anterior, el robot tiene 25 grados de libertad. Eso significa que podemos realizar casi cualquier tipo de movimiento con el robot.

Uno de los requisitos que tenemos que tener en cuenta a la hora de generar un movimiento es la forma en la cual NaoQi acepta la información de esos movimientos. Por ese motivo, cada movimiento que se genera se guarda en un fichero en un formato que explicaremos a continuación.

NaoQi necesita que el formato en el cual le llega la información del movimiento sea en 3 grupos diferenciados. Por un lado, un grupo con los nombres de los motores que se van a mover; por otro lado, un grupo con los ángulos a los que se van a mover los motores y por último, un grupo con los tiempos en los que cada motor va a realizar el movimiento. Estos tres grupos deben tener el mismo número de elementos y el tiempo siempre debe ser incremental.

También es necesario comentar que el componente `Body`, del cual hemos hablado antes, será uno de los componentes que interprete estos ficheros, y de enviarle la información a NaoQi.

En la figura 4.12 podemos ver la interfaz de la herramienta para crear los movimientos para el robot y las funcionalidades que nos ofrece.

- **Load .Mov**. Desde este botón podremos cargar un movimiento ya creado con la extensión .mov. Cuando se intente cargar el movimiento, si éste no sigue las pautas antes definidas, no se cargará el fichero en la herramienta y nos mostrará un error. Por otro lado, si el fichero tiene el formato correcto, el movimiento se cargará en la herramienta y podremos ver: el nombre del movimiento, el número de movimientos que tiene y el número de motores que intervienen.

- **Previous** y **Next**. Estos dos botones nos servirán para poder avanzar o retrocer en el movimiento. Mientras vamos pulsando algunos de estos dos botones, la caja de texto **Mov Position** irá apareciendo la posición del movimiento en la cual nos encontramos. Por otro lado, observaremos como cada vez que pulsamos alguno de estos dos botones todos los *sliders* y las cajas de texto *Time* iran variando, ya que, iran mostrando los valores que tienen en cada posición del movimiento. También

veremos como los checkbox *Do* que pertenezcan al movimiento también se activarán solos.

- **Test Robot**. Este botón nos servirá para poder probar una sola posición de todo el movimiento. Cuando pulsemos este botón, podremos ver como el robot se coloca en la posición que tenemos definida en ese momento.
- **Play**. Mediante este botón se inicia el movimiento y veremos como lo realiza hasta el final.
- **Modify Movement**. Este botón sólo tiene sentido cuando se ha cargado un movimiento previamente. Carga toda la información en la parte de la herramienta de edición de manera que sería como si hubiésemos creado el movimiento en este momento. A partir de que se pulsa el botón podremos modificar los valores de las posiciones: ángulos, tiempos y añadir nuevas posiciones al principio, al final o en medio...
- **Previous** y **Next**. Estos dos botones realizarán las mismas funciones que los botones Previous y Next que están en la zona de visualización de movimientos.
- **Test Robot** y **Play**. Al igual que antes, estos dos botones realizan la misma función que en la parte de visualización de movimientos.
- **Add**. Este botón nos servirá para añadir, siempre al final, una nueva posición dentro del movimiento. Para que se pueda añadir una nueva posicion se deben guardar los ángulos de los motores que están actuando en el movimiento. Si no es así, obtendremos un aviso.
- **Insert**. Este botón es parecido al botón *Add*, pero con este botón podremos insertar una nueva posición donde queramos: al principio, al final o entre medias de otras posiciones. También nos avisará si nos falta algún elemento por guardar.
- **Delete**. Este botón nos servirá para eliminar una posición del movimiento.
- **Finalize**. Mediante este botón definimos el final de un movimiento. Esto es así porque añade después de la posición que se esta editando los valores de una posición definida. Es una posición en la que el robot estará de pie y de una forma estable. Esto nos hará acabar los movimientos de una forma más natural. También conseguiremos que se guarde el fichero en la ruta definida.

- **Save**. Mediante este botón iremos guardando en la ruta deseada el fichero que representará el movimiento. Este botón, a diferencia del anterior, sólo guarda la información, sin añadir ninguna posición más.

- **New .Mov**. Con este botón empezaremos a definir un movimiento nuevo y le indicaremos donde queremos guardarlo.

- **Init Position Robot**. Con este botón haremos que el robot se coloque en una posición definida en la cual el robot estará de pie y de una forma estable. Esto nos vendrá bien para empezar los movimientos en esa posición.

- **Get Value Robot**. Mediante este botón obtendremos todos los valores de los motores en los cuales se encuentra el robot en un momento determinado. Esto hará que todos las barras de desplazamiento modifiquen sus valores.

- **Disable Stiffness**. Cada parte del cuerpo del robot, como se puede observar en la interfaz de la figura 4.12, tiene un *checkbox* al lado con este nombre. Nos servirá para poder desactivar o activar la fuerza de los motores. Gracias a poder realizar esta función, podremos desactivar la fuerza del motor de la parte del robot que necesitemos para colocarlo a nuestro gusto.

- **Do**. Este *checkbox* sólo se activará cuando se cree un nuevo movimiento. También se puede observar que aparece en cada una de las partes del cuerpo. Se activará para definir que esa parte del cuerpo interviene en el movimiento.

- **Send to Robot**. Como los dos *checkbox* anteriores, éste también aparece en las diferentes partes del cuerpo. Cuando se activa este checkbox lo que conseguimos es que al mover cualquiera de las barras de desplazamiento la información de ese motor se le envíe al robot al momento. Sólo se le envía la información del motor que cambiemos su valor.

- **Get Value**. Este botón aparece en cada una de las partes del robot también. Nos servirá para obtener los valores de esa parte del cuerpo en el momento que lo pulsemos. Gracias a poder cambiar la posición de cualquier parte del cuerpo desactivando la fuerza de sus motores, podemos, por ejemplo, mover un brazo hacía arriba. Cuando volvemos a activar el motor para que se quede fijo pulsar este botón y obtener los valores de sus motores.

- **Save Value**. Este es otro botón que aparece en cada parte del cuerpo. Es el que se encarga de guardar la información de esa parte del cuerpo dentro del movimiento.

- **Sliders**. Son cada una de las barras en las que podemos establecer el ángulo de cada uno de los motores.
- **Time**. Son los segundos en los cuales se va a ejecutar esa posición. Es importante que siempre el tiempo sea incremental.

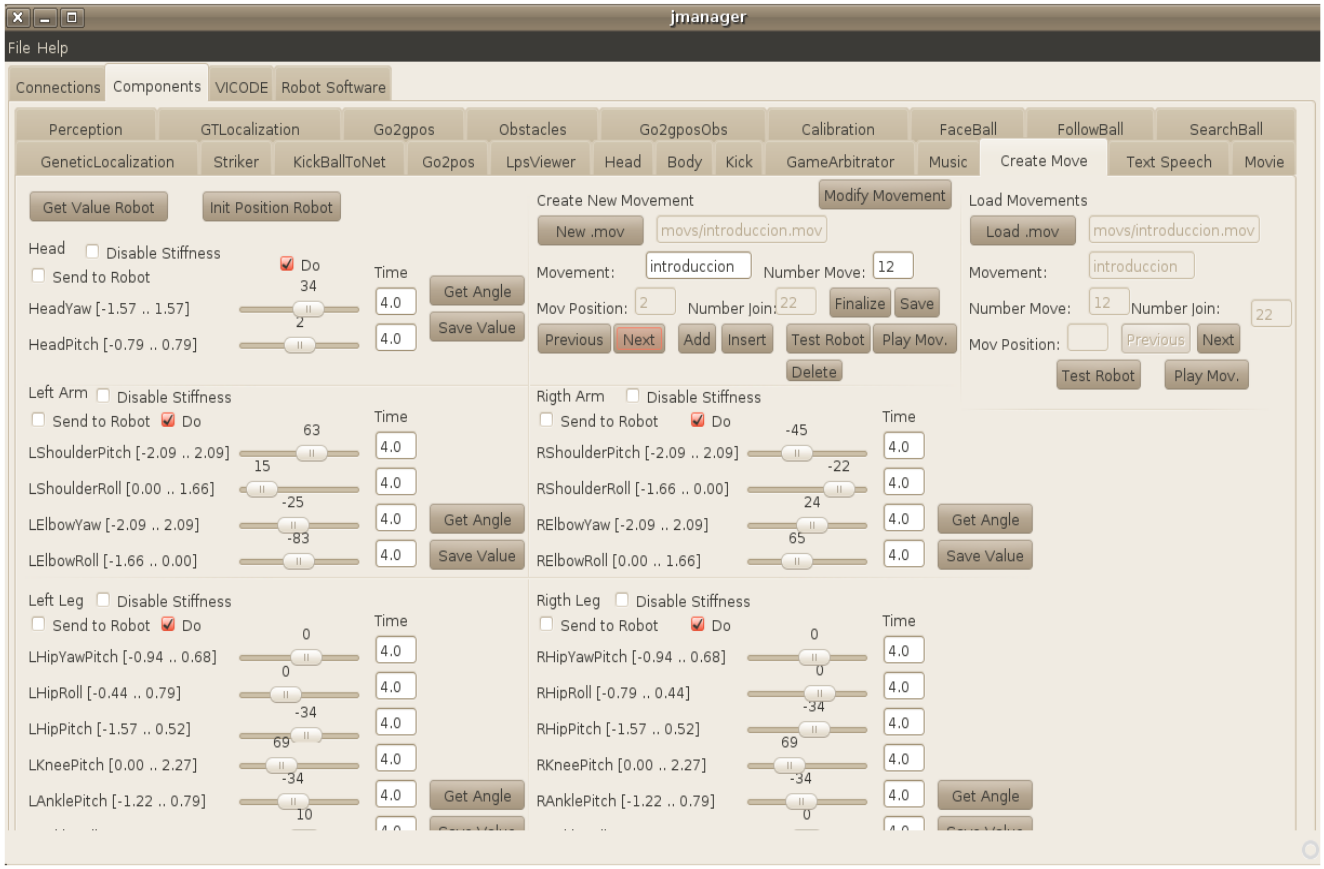


Figura 4.12: Herramienta Create Movie

A continuación, vamos a explicar uno de los ficheros generados con esta herramienta para la realización de un movimiento. El fichero que se puede ver a continuación, está representando un movimiento que tan sólo utliza los brazos. El movimiento que está haciendo es subir y bajar los brazos 10 veces. Podemos ver que el fichero empieza con el nombre del movimiento, *brazo1*, seguido de el número de posiciones que tendrá el movimiento, *15*. Y por último, aparece la duración del movimiento, *28,5* segundos.

En la siguiente línea aparecen el nombre de cada uno de los motores que intervienen en el movimiento, mostrados en la figura 4.13.

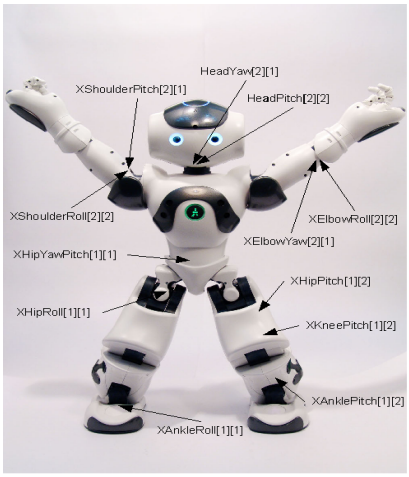


Figura 4.13: Motores de Nao

Las siguientes 7 líneas que aparecen corresponden cada línea a un motor, es decir, la primera línea le pertene al motor *LShoulderPitch*, la segunda al motor *LShoulderRoll*, y así sucesivamente. Dentro de cada una de estas 7 líneas están los ángulos en los cuales se tendrán que colocar los motores en cada una de las 15 posiciones.

Por último, hay otras 7 líneas que están relacionadas con los motores de la misma manera que las 7 anteriores. Lo que nos indican los valores de cada línea son los segundos en los cuales los motores se tienen que colocar en los ángulos correspondientes.

Por ejemplo, *LShoulderPicth* en el segundo 1 tiene que estar en el ángulo 1.32. En el segundo 2.5 tendrá que estar en el -0.76, y así sucesivamente.

```
brazos1 15 32.5
LShoulderPitch LShoulderRoll LElbowYaw LElbowRoll RShoulderPitch RShoulderRoll RElbowYaw
1.32 -0.76 1.06 -0.65 0.97 -0.66 0.97 -0.66 0.97 -0.65 0.97 0.97
-0.65 0.97 -0.66 0.97 -0.66 0.97 -0.65 0.97 0.28 0.15 0.16 0.15 0.15 0.15 0.15
0.79 1.27 1.27 1.27 1.27 1.27 1.27 1.27 1.27 1.27 1.27 1.27 1.27 1.27 1.27
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
1.53 -0.81 1.12 -0.61 1.05 -0.64 0.97 -0.64 0.97 -0.64 0.76 0.76 -0.64 0.97 -0.64
-0.34 -0.26 -0.27 -0.27 -0.27 -0.27 -0.27 -0.27 -0.27 -0.27 -0.27 -0.27 -0.27 -0.27 -0.27
0.2 0.18 0.18 0.18 0.18 0.18 0.18 0.18 0.18 0.18 0.18 0.18 0.18 0.18 0.18
1 2.5 4.5 6.5 8.5 10.5 12.5 14.5 16.5 18.5 20.5 22.5 24.5 26.5 28.5
1 2.5 4.5 6.5 8.5 10.5 12.5 14.5 16.5 18.5 20.5 22.5 24.5 26.5 28.5
1 2.5 4.5 6.5 8.5 10.5 12.5 14.5 16.5 18.5 20.5 22.5 24.5 26.5 28.5
1 2.5 4.5 6.5 8.5 10.5 12.5 14.5 16.5 18.5 20.5 22.5 24.5 26.5 28.5
1 2.5 4.5 6.5 8.5 10.5 12.5 14.5 16.5 18.5 20.5 22.5 24.5 26.5 28.5
1 2.5 4.5 6.5 8.5 10.5 12.5 14.5 16.5 18.5 20.5 22.5 24.5 26.5 28.5
1 2.5 4.5 6.5 8.5 10.5 12.5 14.5 16.5 18.5 20.5 22.5 24.5 26.5 28.5
```

Capítulo 5

Experimentos y Pruebas

En este apartado vamos a explicar y a mostrar ejemplos de las pruebas que hemos ido realizando durante todo el proyecto para validar cada una de sus partes, así como probar su funcionalidad global para las sesiones de terapia. Cada prototipo que se ha terminado requería unas determinadas pruebas y comprobaciones para poder pasar al siguiente.

5.1. Pruebas unitarias

Estas pruebas están destinadas a la evaluación y validación de los componentes software durante el desarrollo.

5.1.1. Music

Una vez finalizada la interfaz del componente `Music` y acabado también el desarrollo de éste, las pruebas que se han llevado a cabo han sido siempre usando el interfaz, ya que, es el que nos permite modular el componente desde fuera.

1. Activar y desactivar el componente mediante el checkbox.
2. Reproducir un fichero que existe pulsando el botón *Play*.
3. Intentar cargar un fichero de música que no existe.
4. Parar la reproducción del fichero de música pulsando el botón *Stop*.

Una vez finalizadas las pruebas en las cuales hemos podido comprobar que el componente por si sólo funciona perfectamente podemos avanzar al siguiente prototipo.

5.1.2. TextSpeech

Una vez finalizada la interfaz del componente `TextSpeech` y acabado también el desarrollo de éste, las pruebas que se han llevado a cabo han sido siempre usando el interfaz, ya que, es el que nos permite modular el componente desde fuera.

1. Activar y desactivar el componente mediante el checkbox.
2. Elegir en el desplegable un idioma que no estaba configurado en el robot.
3. Elegir en el desplegable un idioma que si estaba configurado en el robot.
4. Subir y bajar el volumen.
5. Elegir en el desplegable un tipo de voz que no estaba configurada en el robot.
6. Elegir en el desplegable un tipo de voz que si estaba configurada en el robot.
7. Rellenar la caja de texto con varias frases.
8. Obtener los valores del robot pulsando el botón *Get Value*.

Una vez finalizadas las pruebas en las cuales hemos podido comprobar que el componente por si sólo funciona perfectamente podemos avanzar al siguiente prototipo.

5.1.3. LedsControl

Como se ha comentado en la explicación del componente, este componente no tiene interfaz. Por lo que las pruebas se han llevado a cabo modulandolo desde el interfaz del componente final `Movie`.

1. Activar todos los leds automáticamente.
2. Desactivar todos los leds.
3. Activar los leds de los ojos de un color concreto.
4. Desactivar los leds de los ojos.
5. Activar los leds de los pies de un color concreto.
6. Desactivar los leds de los pies.

7. Activar el leds del pecho de un color concreto.
8. Desactivar el leds del pecho.
9. Activar los leds de las orejas.
10. Desactivar los leds de las orejas.

5.1.4. Body

Una vez finalizada la interfaz del componente `Body` y acabado también el desarrollo de éste, las pruebas que se han llevado a cabo han sido siempre usando el interfaz, ya que, es el que nos permite modular el componente desde fuera.

1. Activar y desactivar el componente mediante el checkbox.
2. Pulsar los botones, *Walk*, *RotateLeft*, *RotateRight*, *Backword*, *Stop*.
3. Pulsar *doMove*.

Una vez finalizadas las pruebas en las cuales hemos podido comporobar que el componente por si sólo funciona perfectamente podemos avanzar al siguiente prototipo.

5.1.5. Movie

Una vez que todos los componentes han sido desarrollados y probados y una vez finalizada la interfaz del componente `Movie` y acabado también el desarrollo de éste, las pruebas que se han llevado a cabo han sido siempre usando el interfaz ya que es el que nos permite modular el componente desde fuera.

1. Activar y desactivar el componente mediante el checkbox.
2. Cargar un guión que existe pulsando el botón *Load .Mov*.
3. Cargar un guión que no existe en el robot.
4. Crear un nuevo guión pulsado el botón *Create .Mov*.
5. Salvar un guión pulsando el botón *Save*.
6. Enviar un guión al robot pulsando el botón *Send To Robot*.

7. Arrancar un guión pulsando el botón *Play*.
8. Avanzar y retroceder a un *Breakpoint*.

Una vez que hemos finalizado estas pruebas podemos decir, que todos los componentes funcionan correctamente y podemos pasar a realizar dos experimentos más para completar la expiremetación de este proyecto.

5.1.6. Create Movie

En este apartado vamos a explicar las pruebas que se han llevado a cabo para la herramienta de generación de movimientos.

1. Pulsar el botón *Load .mov*.
2. Pulsar los botones *Previous* y *Next*.
3. Pulsar el botón *Test Robot*.
4. Pulsar el botón *Play Mov*.
5. Pulsar el botón *Modify Movement*.
6. Pulsar el botón *New mov*.
7. Pulsar el botón *Get Value Robot*.
8. Pulsar el botón *Init Position Robot*.
9. Pulsar los botones *Previous* y *Next* de edición.
10. Pulsar el botón *Add*.
11. Pulsar el botón *Insert*.
12. Pulsar el botón *Test Robot* de edición.
13. Pulsar el botón *Play Mov* de edición.
14. Pulsa el botón *Finalize*.
15. Pulsar el botón *Save*.
16. Comprobación de los *sliders*

17. Pulsar cualquier botón *Get Value*.

18. Pulsar cualquier botón *Save Value*.

19. Activar y desactivar el *checkbox Disable Stiffness*.

20. Activar y desactivar el *checkbox Do*.

21. Activar y desactivar el *checkbox Send to Robot*.

5.2. Experimentos

En este apartado vamos a comentar la sesión previa que se llevo a cabo en Diciembre del 2009 en el Centro de Alzhéimer de la Fundación Reina Sofía[1] para comprobar la viabilidad de la realización de este proyecto. Además, explicaremos como deberían realizarse dos sesiones de terapia.

Como hemos comentado, en Diciembre del 2009 se realizó una prueba prevía para valorar la vialibidad de la realización de este proyecto y comprobar si los enfermos aceptaban al robot. En esta prueba se usaron el robot canino Aibo (figura 1.3(a)) y el robot Nao (figura 1.10(c)). En la figura 5.1 se puede observar momentos de la terapia.

(a) (b)

Figura 5.1: Terapia con robots

Los resultados obtenidos en la prueba fueron expectaculares, desde el punto de vista de los terapeutas y expertos en la enfermedad, sobre todo cuando se uso el robot Nao. Al robot Nao se le programó para cantar y bailar. Al principio de la sesión había enfermos que

[1] http://www.fundacionreinasofia.es/

eran totalmente pasivos, pero según iba avanzando la sesión muchos de ellos se encontraban cantando y aplaudiendo. Según comentaron los expertos no habían visto estos resultados ni con el robot *Paro*, con el que cuentan en este centro.

A continuación vamos a explicar dos sesiones de terapia con enfermos de Alzhéimer, para explicar como se deberían realizar usando nuestro sistema. Estas sesiones están supervisadas por expertos en la matería.

5.2.1. Primera sesión

La primera sesión es una sesión basada en el lenguaje. Será una sesión, en la cual, jugaran con el robot para adivinar los antónimos de las palabras que les diga el robot.

Desarrollo de la sesión: Se les saludará a cada participante de forma individual. Tras esto les presentaremos a nuestro amigo y a continuación el robot comenzará a interactuar con ellos: Buenos días a todos, hoy tengo muchas ganas de hablar así que espero que todos vosotros tengáis ganas de escucharme. Lo que no es blanco es . . . negro.

- RICO - POBRE
- GRANDE - PEQUEÑO
- ANCHO - ESTRECHO
- LARGO - CORTO
- GORDO - FLACO
- VIEJO - JOVEN
- MOJADO - SECO
- BLANCO - NEGRO
- GUAPO - FEO
- ALTO - BAJO
- CLARO - OSCURO
- ARRIBA - ABAJO
- DENTRO - FUERA

- ABIERTO - CERRADO
- VERANO - INVIERNO
- BUENO - MALO

A continuación, vamos a mostrar el guión que representa esta sesión:

```
wait time 2
breakpoint
talk file /home/nao/movie/text/presentacion.txt
wait task talk
mov introduccion
wait time 1
talk text Buenos días a todos, ,hoy tengo muchas ganas de hablar así que
  espero que todos vosotros tengáis ganas de escucharme. Vamos a
  jugar a un juego, , para conocer los antónimos. wait task mov
breakpoint
wait task talk
talk text Lo que no es Blanco , , es?
wait press left
led 0 2
talk text Negro
wait press left
breakpoint
talk text Vamos a por la siguiente!
wait task talk
talk text Lo que no es Rico , , es?
wait press left
led 0 2
talk text Pobre
wait press left
breakpoint
talk text Vamos a por la siguiente!
wait task talk
talk text Lo que no es grande , , es?
wait press left
led 0 2
talk text pequeño
wait press left
breakpoint
talk text Vamos a por la siguiente!
wait task talk
talk text Lo que no es ancho , , es?
```

```
wait press left
led 0 2
talk text estrecho
wait press left
breakpoint
talk text Vamos a por la siguiente!
wait task talk
talk text Lo que no es largo , , es?
led 0 2
wait press left
talk text corto
wait press left
breakpoint
talk text Vamos a por la siguiente!
wait task talk
talk text Lo que no es  gordo , , es?
wait press left
led 0 2
talk text flaco
wait press left
breakpoint
talk text Vamos a por la siguiente!
wait task talk
talk text Lo que no es viejo , , es?
wait press left
led 0 2
talk text joven
wait press left
breakpoint
talk text Vamos a por la siguiente!
wait task talk
talk text Lo que no es mojado , , es?
wait press left
led 0 2
talk text seco
wait press left
breakpoint
talk text Vamos a por la siguiente!
wait task talk
talk text Lo que no es guapo , , es?
wait press left
led 0 2
talk text feo
```

```
wait press left
breakpoint
talk text Vamos a por la siguiente!
wait task talk
talk text Lo que no es alto , , es?
wait press left
led 0 2
talk text bajo
wait press left
breakpoint
talk text Vamos a por la siguiente!
wait task talk
talk text Lo que no es claro , , es?
wait press left
led 0 2
talk text oscuro
wait press left
breakpoint
talk text Vamos a por la siguiente!
wait task talk
talk text Lo que no es arriba , , es?
wait press left
led 0 2
talk text abajo
wait press left
breakpoint
talk text Vamos a por la siguiente!
wait task talk
talk text Lo que no es dentro , , es?
wait press left
led 0 2
talk text fuera
wait press left
breakpoint
talk text Vamos a por la siguiente!
wait task talk
talk text Lo que no es abierto , , es?
wait press left
led 0 2
talk text cerrado
wait press left
breakpoint
talk text Vamos a por la siguiente!
```

```
wait task talk
talk text Lo que no es verano , , es?
wait press left
led 0 2
talk text invierno
wait press left
breakpoint
talk text Vamos a por la siguiente!
wait task talk
talk text Lo que no es bueno , , es?
wait press left
led 0 2
talk text malo
wait press left
breakpoint
talk text Estoy muy contento por haber estado jugando y hablando con vosotros. ,
  Espero volver a veros a todos el próximo día. Adios a todos.
```

5.2.2. Segunda sesión

La segunda sesión es una sesión basada de fisioteparia, donde a la vez que da la orden de la realización del ejercicio, realiza la acción de forma pausada.

Desarrollo de la sesión: Se les saludará a cada participante de forma individual. Tras esto les presentaremos a nuestro amigo. A continuación el Robot comenzará a interactuar con ellos: Buenos días a todos, hoy tengo muchas ganas hacer ejercicio así que espero que todos me acompañéis para que nos pongamos en forma.

Vamos a comenzar por el cuello: Mover la cabeza hacia delante y detrás (como diciendo 'si' con la cabeza). 1 2 3 4 5 6 7 8 9 10

A continuación, vamos a mostrar el guión que representa esta sesión:

```
wait time 2
breakpoint
talk file /home/nao/movie/text/presentacion.txt
wait task talk
mov introduccion
wait time 1
talk text Buenos días a todos, hoy tengo muchas ganas hacer ejercicio
así que espero que todos me acompañéis para que nos pongamos en forma.
wait task talk
```

```
mov diquesi
talk text 1 2 3 4 5 6 7 8 9 10
wait task mov
```

El fichero generado a partir de la herramienta de JManager del movimiento *diquesi* es el siguiente:

```
si 17 32.5
HeadYaw HeadPitch
-0.03 -0.04 -0.04 -0.04 -0.04 -0.04 -0.04 -0.04 -0.04 -0.04 -0.04 -0.04 -0.04 -0.04 -0.04 -0.04
0.02 -0.67 0.51 -0.67 0.51 -0.67 0.51 -0.67 0.51 0.51 -0.67 0.51 -0.67 0.51 -0.67 0.51 -0.67
1 2.5 4.5 6.5 8.5 10.5 12.5 14.5 16.5 18.5 20.5 22.5 24.5 26.5 28.5 30.5 32.5
1 2.5 4.5 6.5 8.5 10.5 12.5 14.5 16.5 18.5 20.5 22.5 24.5 26.5 28.5 30.5 32.5
```

Capítulo 6

Conclusiones y Trabajos futuros

Una vez que ya hemos explicado el funcionamiento del software para este comportamiento, las pruebas realizadas y los avances en la interfaz de JManager y en la arquitectura BICA, podemos acabar esta memoria resumiendo algunos puntos principales de este proyecto y definiendo algunas posibles mejoras.

6.1. Conclusiones

Como conclusión principal podemos decir que se ha conseguido extender la arquitectura BICA con una funcionalidad capaz de realizar lo que se proponía en el proyecto: un sistema para poder generar coreografías para que las ejecutase un robot humanoide. La generación de estas coreografías se ha conseguido que sea de una forma fácil e intuitiva, tal y como se requería en los requisitos. El comportamiento se ha realizado usando el robot humanoide real, Nao, como escenario de pruebas ya que era uno de los requisitos necesarios.

A continuación vamos a enumerar todo el software que hemos realizado:

- Diseño de el lenguaje de coreografías. Se ha creado un lenguaje fácil de entender para poder diseñar las coreografías para el robot.
- Herramienta de generación de movimientos para el robot y su interfaz. Se ha desarrollado una herramienta con la cual la generación de movimientos con el robot es sencilla e intuitiva.
- Componente `Music` y su interfaz. Se ha diseñado un interfaz desde el cual somos capaces de reproducir ficheros de música en el robot.

- Componente `TextSpeech` y su interfaz. Se ha diseñado un interfaz desde el cual somos capaces de que el robot nos hable.
- Componente `LedsControl`. Se ha diseñado un componente que nos facilita crear juegos de luces.
- Implementación y realización de las pruebas. Se han realizado todas las pruebas necesarias para cada uno de los componentes y sus interfaces. Hemos comprobado que todo funciona correctamente, tanto individual como colectivamente.
- Implementación de las coregorafías para las sesiones de terapia propuestas por el terapeuta.

Otro de los aspectos que es importante destacar es la labor de integración que se ha llevado a cabo en este proyecto. Integrando el software de este proyecto en una arquitectura y una interfaz ya creada. Esto ha llevado a que exista un periodo de aprendizaje de la arquitectura.

Entre los conocimientos que me han aportado este proyecto se pueden destacar conocer el funcionamiento de la arquitectura BICA, que usa el Grupo de Robótica para el desarrollo de las aplicaciones para la participación en la Robocup. También hay que destacar el aprendizaje con la plataforma NaoQi, el conocimiento de la mayoría de sus módulos y de su funcionamiento.

Otro de los requisitos que se ha cumplido ha sido la ampliación de aplicaciones en la herramienta JManager, ya que, por cada componente que hemos desarrollado, éste ha ido acompañado de su aplicación particular dentro de la herramienta.

Destacar que se han cumplido todos los objetivos que se buscaban con la realización de este proyecto y que se han podido realizar todas las pruebas deseadas.

Por último, decir que nos hubiese gustado realizar las pruebas oportunas en un hospital pero debido a los plazos de tiempo no se ha podido. Las pruebas están planificadas y se realizaran para confirmar que todo funciona y que los pacientes reaccionan como se han comentado en la prueba piloto que se realizó.

6.2. Posibles trabajos futuros

En este último apartado de la memoria vamos a comentar algunos de los posibles trabajos futuros que podrían hacerse.

- Integrar el interfaz en un móvil con Android[1] para que su usabilidad sea aún más sencilla y manejable.
- Mejorar la herramienta de generación de movimientos incluyendo una línea de tiempo.
- Mejorar la herramienta de generación de coreografía incluyendo un botón por cada uno de los comportamientos y conseguir que sea todavía más fácil generar las coreografías.
- Realizar las pruebas en el hospital.

[1] http://es.wikipedia.org/wiki/Android

Bibliografía

[Asimov, 1950] Isaac Asimov. *Yo, Robot.* Doubleday, 1950.

[Bernardo Cascales Salinas, 2000] Pascual Lucas Saorín Bernardo Cascales Salinas. *Latex, una imprenta en sus manos.* AD, 2000.

[Capek, 1920] Karel Capek. *Rossumovi univerzální roboti.* Desconocida, 1920.

[Ceballos Sierra, 2009] Francisco Javier Ceballos Sierra. *Enciclopedia del lenguaje C++.* RA-MA, 2009.

[Francisco Martín, 2009] Carlos E. Agüero y José María Cañas Francisco Martín. Follow ball behavior for an humanoid soccer player. *Universidad Rey Juan Carlos*, 2009.

[Francisco Martín, 2010] José María Cañas y Eduardo Perdices Francisco Martín, Carlos Agüero. Humanoid soccer player design. *Vladan Papic*, 2010.

[Heffelfinger, 2008] David Heffelfinger. *Java EE 5 Development with NetBeans.* Packt Publishing, 2008.

[Macchiavello, 2008] Tatiana Macchiavello. Robótica. *Monografías*, 2008.

[Mejías, 2004] Raúl Benítez Mejías. Sistema de localización basado en la detección de segmentos para un robot móvil. *Universidad Rey Juan Carlos-PFC*, 2004.