Manual de R-CODE, v2.1

por Ricardo A. Téllez (r_tellez@ouroboros.org)

23rd July 2005

Este manual es email-ware. Esto significa que si usas este manual debes enviarme un email diciendome lo que te parece y cualquier sugerencia que consideres. También agradeceré que me indiques cualquier fallo que hayas podido detectar para que pueda corregirlo en versiones posteriores. A mi me llevó mucho tiempo escribir este manual, pero a ti sólo te llevará un minuto enviar el email.

Contents

Chapter 1

Introducción al R-CODE

El conjunto de utilidades llamado R-CODE SDK de Sony forman un entorno de programación para el robot Aibo en el cual se utiliza un lenguaje de script llamado R-CODE. Por lenguaje de script se entiende un lenguaje de programación de alto nivel formado por comandos simples, y cuyos programas se graban en un fichero de texto sin formato. Estos comandos permiten al programador enviar órdenes al robot de una forma parecida al uso de macros, sin necesidad de utilizar un lenguaje complicado de bajo nivel como C++ (como por ejemplo ocurre con los programas de OPEN-R para Aibo). Los lenguajes de script tienen la ventaja de ser fáciles de aprender y usar, y no requiren compilación. Por contra, tienen la desventaja de que no permiten un control preciso a la hora de realizar ciertas tareas. El R-CODE SDK está compuesto por los ficheros Redist7, RTool y NSMAuth, siendo Redist7 el fichero principal que proporciona el sistema base para el desarrollo en el lenguaje R-CODE.

Un ejemplo de programa en R-CODE puede verse aquí:

```
:1000
PLAY:ACTION:STAND
IF:Face:=:1:THEN
WAIT
PLAY:ACTION:CLIFF_DETECT_OFF
WAIT
```

R-CODE permite la programación de acciones bastante complicadas, como por ejemplo el caminar o el bailar, con tan solo unas pocas líneas de código. Una forma de ver R-CODE es como un conjunto de macros y OPEN-R como el código de esas macros. Con el uso de macros no puede obtenerse más precisión que aquella que haya sido codificada en el código interno de las mismas (en OPEN-R). Es una buena idea utilizar R-CODE cuando no sea necesario un control muy preciso del robot, y lo que se desea es realizar acciones, movimientos o respuestas predefinidas. No obstante, no debe confundirse R-CODE con algo simple, ya que permite la implementación de comportamientos muy complejos.

Para poder utilizar R-CODE con Aibo son necesarias dos cosas: en primer lugar, un *memory stick* propiamente configurado, y en segundo lugar, un programa escrito en lenguaje R-CODE. Para configurar el memory stick será necesario disponer de uno vacío, en el cual se grabarán diversos ficheros proporcionados por Sony (los ficheros de Redist7[1]). Estos ficheros forman el llamado *sistema base* y forman parte del R-CODE SDK. Una vez el memory stick ha sido preparado con el sistema base, es posible entonces transferir el programa R-CODE diseñado por el usuario al memory stick y ejecutarlo después en Aibo.

Aunque el R-CODE SDK contiene varios ficheros de uso específico con sistemas Microsoft Windows (los ficheros RTool y NsmAuth), el desarrollo de programas en R-CODE puede realizarse con cualquier sistema operativo que disponga de un editor de texto, ya que los programas de R-CODE son tan sólo ficheros de textos que no necesitan compilación. Los ficheros RTool y NsmAuth proporcionan utilidades complementarias que no será necesario utilizar en la mayoría de los casos.

El listado siguiente contiene una descripción de lo que puede realizarse con R-CODE:

- Poner a Aibo en sus tres posturas básicas: sentado, de pie y durmiendo.
- Hacer a Aibo andar, girar sobre sí mismo, chutar, tocar, mover la cabeza y seguir con la mirada la pelota rosa.
- Hacer a Aibo buscar la pelota rosa, una cara o el hueso rosa (AIBOne).
- Hacer a Aibo reconocer alguno de los 53 comandos verbales, 35 sonidos y 68 tonos musicales
- Ejecutar 600 contenidos, entre los que destacan movimientos, sonidos y patrones de luces LEDs
- Ejecutar movimientos creados utilizando el AIBO Motion Editor
- Ejecutar sonidos WAV y MIDI creados por el usuario
- Ejecutar patrones de luces LED creados por el usuario
- Adquirir datos acerca de la detección de obstáculos (distancias), así como datos de los sensores de la cabeza. boca y espalda.
- Utilizar secuencias de control que incluyen sentencias *IF* y *FOR*
- Ejecutar subrutinas
- Utilizar variables, aunque sólo enteros de 32 bits
- Realizar operaciones aritméticas

[1] Redist7 contiene los ficheros del sistema base de R-CODE para el modelo ERS-7 de Aibo. Para otros modelos de Aibo es necesario utilizar otros ficheros del sistema base, que también son proporcionados por Sony. Este libro se concentra en el sistema R-CODE para ERS-7.

- Utilizar una pila y aplicar sobre ella operaciones de *PUSH* y *POP*
- Utilizar las capacidades wireless para acceder a una consola de control desde un ordenador remoto. Esta opción permite el envio de comandos remotos a Aibo y el debugado de programas R-CODE

Chapter 2

Instalación y configuración

2.1 Descripción

El R-CODE SDK se compone de las siguientes utilidades:

- *Redist7*, compuesto de un conjunto de ficheros que forman el sistema base a instalar en el memory stick. Es necesario tener el sistema base en el memory stick para poder ejecutar los programas de R-CODE en Aibo. Redist7 contiene versiones en Japonés y en Inglés del sistema base, por lo que el programador deberá seleccionar cual de ellas desea utilizar. El lenguaje seleccionado afecta principalmente a las órdenes habladas y a las respuestas que Aibo pueda dar.
- *RTool*, es una aplicación para sistemas Microsoft Windows que convierte ficheros MTN, WAV, MID y LED (estos son, ficheros que describen movimientos, sonidos y patrones de luces del robot), en ficheros de formato ODA, que es un formato reconocible por R-CODE y susceptible de ser utilizado dentro de un programa (ver el capítulo 12 para más información).
- *NsmAuth*, es una librería preparada para ser enlazada (linkada) con programas Windows de C++ que deseen conectarse a la consola de control remoto de R-CODE y utilizar autentificación.

2.2 Instalación de R-CODE

En primer lugar será necesario descargar de Internet los ficheros de R-CODE. Estos pueden localizarse en la página de Sony destinada al OPEN-R (openr.aibo.com). Será necesario descargar los ficheros Redist7_verx.zip, Rtool_verx.zip y NsmAuth_verx.zip, aunque en este caso tan solo el primer fichero será necesario, ya que los otros dos corresponden a entornos mas complicados que serán tratados más adelante.

Preparación del memory stick

Tras descargar los archivos necesarios, se procederá a instalar el sistema base en el memory stick. De esta manera, el stick estará preparado para acojer los programas en R-CODE que desarrolle el programador. Para realizar la instalación del sistema base, primero hay que descomprimir el fichero Redist7_verx.zip, lo que generará un conjunto de ficheros y directorios. Tras la descompresión, deberá compiarse el directorio Redist7/Eng/OPEN-R obtenido en un memory stick vacío (para un entorno R-CODE en inglés. Copiar el directorio Redist7/Jap/OPEN-R para un entorno en Japonés). En este punto el stick ya está preparado para ejecutar programas de R-CODE. Consulte el apéndice A sobre como configurar el lector/grabador de memory sticks bajo Windows o Linux.

Ejecución de un programa en R-CODE

Ahora ya es posible crear un programa en R-CODE utilizando cualquier editor de textos. Una vez finalizado el programa, éste deberá ser grabado con el nombre R-CODE.R (no puede tener otro nombre mas que ese) en el directorio /OPEN-R/APP/PC/AMS/ del memory stick. A continuación el memory stick ya puede ser insertado en el robot. Al activar a Aibo, se ejecutará automáticamente el programa creado.

Configurando la consola inalámbrica

La consola inalámbrica permite al usuario el envío directo de comandos R-CODE al robot a través de una sesión de telnet desde un ordenador remoto que tenga acceso a la red inalámbrica. Esto tiene varias ventajas como la de permitir el debugado de programas online, o el testeo de rutinas y comandos. Para configurar la consola hay que realizar los siguientes pasos:

1. Configurar la red inalámbrica. Consulte el apéndice B sobre cómo realizar la configuración.
2. Deshabilitación de la autentificación en R-CODE. Deshabilitando la autentificación no será necesario introducir un identificador de usuario ni una contraseña cuando se conecte a la consola por telnet. Para realizar esta deshabilitación se deberá borrar el fichero /OPEN-R/APP/DATA/P/OWNER.TXT del memory stick, y crear en él el fichero /OPEN-R/APP/PC/AMS/NOAUTH.CFG de tamaño cero.
3. Insertar el memory stick en el robot e iniciarlo.
4. Conectar con el robot desde un ordenador remoto usando telnet. Las conexiones deben realizarse en el puerto 21002, por lo que el comando de conexión debería ser *telnet IP_del_Aibo 21002*. Si la conexión tiene éxito, la pantalla mostrará una cabecera producida por R-CODE conteniendo diversa información, tras la cual aparecerá el signo de invitación a introducir un comando (ver figura 1).

5. En este momento es posible enviar comandos R-CODE directamente a Aibo con tan solo teclearlos en la consola.

6. También es posible enviar todo un programa desde la consola para que éste sea ejecutado por Aibo. Para ello primero es necesario teclear el comando *EDIT* en la consola R-CODE, el cual indica que a continuación va a introducirse todo un programa y no un conjunto separado de instrucciones. Tras teclear todo el programa en la misma consola, se introduce el comando *END*, indicando que el programa ha sido finalizado. Para ejecutar el nuevo programa introducido, bastará con teclear en la consola el comando *RUN*. Hay que hacer notar que cualquier programa enviado al robot de esta manera sustituye el actual programa en memoria del robot, pero no el programa que pueda haber en el memory stick insertado en el robot.

7. Para desconectar de la consola se envía el comando *@DISS*.

```
/Users/rick/Documents/instant.tif not found!
```

2.3 Consideraciones generales

Antes de comenzar a programar con R-CODE es necesario tener en cuenta las siguientes consideraciones generales:

1. Los scripts de R-CODE están compuestos de líneas de texto que especifican comandos. Los comandos se forman con palabras separadas por el símbolo de dos puntos (:). En las palabras sólo pueden utilizarse caracteres ASCII y guiones bajos, y una línea de comando no puede ser mas larga de 127 bytes, incluyendo comentarios y el retorno de carro. Los caracteres de indentación, espacios y tabuladores al principio de la línea, son ignorados.

2. Las líneas que comienzan con el símbolo de dos puntos (:) se consideran etiquetas. Las etiquetas son utilizadas en el código para especificar puntos de salto, a los cuales se accede desde el programa mediante el comando *GO*. Las líneas que comienzan con un caracter que no es un alfanumérico o dos puntos, son tratadas como comentarios. Un comentario puede ser también especificado con la doble barra (//). Todo lo que venga después de una doble barra es tratado como comentario.

3. R-CODE es sensible a las mayusculas y minúsculas. Esto significa que *Valor* y *VALOR* son variables diferentes. En general, las palabras completamente en mayúsculas están reservadas para el uso interno de R-CODE, por lo que conviene no utilizar palabras completamente en mayúsculas para definir variables de usuario, ya que podrían provocar confusión y errores. Por ejemplo, *Valor* y *valor* son buenos nombres de variables, pero *VALOR* no lo es.

4. Las palabras que empiezan por el símbolo mas (+), menos (-) o un número son tratadas como números. Los números que empiezan por 0x o 0X son tratados como números hexadecimales. Los números que empiezan por 0o o por 0O son tratados como números en octal. Por último, los números que empiezan por 0b o por 0B son tratados como números binarios.

5. Respecto a la definición de números, tan solo pueden utilizarse números enteros de un máximo de 32 bits (en realidad 31 bits más el signo).

Chapter 3

Programando en R-CODE

Hablando en términos generales, un programa de R-CODE consiste en una serie de commandos que harán que el robot escuche, se mueva y realice acciones. La mayoría de las acciones y reconocimientos están ya creadas dentro de la especificación de R-CODE con lo que el programador tan sólo tiene que hacer uso de ellas. A continuación se describen los elementos que componen un programa de R-CODE.

Programa

Cualquier programa de R-CODE es una lista de comandos para el robot Aibo. Los programas en este lenguaje se parecen mucho a los programas en Basic, e implementan todas las funciones típicas de un lenguaje de este tipo como condicionales, subrutinas, control del flujo, uso de variables, etc, pero además, R-CODE implementa el uso de la pila, de una manera similar a la del lenguaje ensamblador (aunque en este caso la pila es FIFO en lugar de LIFO). En el capítulo 4 podrá encontrarse una lista exhaustiva de los comandos disponibles en R-CODE.

Normalmente, un programa de R-CODE es generado utilizando un editor de texto ASCII. El programa es creado usando el editor y una vez finalizado, es grabado en el memory stick (directorio /OPEN-R/APP/AMS/) con el nombre R-CODE.R. Entonces el memory stick es insertado en Aibo y el programa ejecutado cuando el robot es activado. También es posible la creación de un programa directamente en la memoria de Aibo utilizando la consola wireless, siendo esta aproximación mucho más flexible y rápida a la hora de depurar los programas y encontrar sus errores.

Variables

R-CODE incluye de serie un conjunto de variables que permiten al programador identificar el estado del robot. Una descripción completa de todas las variables disponibles se encuentra en el capítulo 4. De entre las variables, algunas corre-

sponden al valor de los estados internos del robot (modo de operación, detección de obstáculos activada, etc...) y otras corresponden a valores de los sensores del robot. El programador puede obtener en cada instante el valor de estas variables y actuar en consecuencia. Por ejemplo, existe una variable llamada *Face*, la cual indica cuando Aibo ha detectado una cara (si ha sido detectada, la variable toma el valor 1). El programador podría utilizar esta variable para detectar si Aibo se encuentra de cara con una persona. Existen otras muchas variables que muestran por ejemplo el estado de los sensores de los pies, la detección de la pelota o la del AIBOne (hueso de Aibo). El programador puede también crear sus propias variables y utilizarlas para realizar cálculos o pasarlas como parámetros a subrutinas (mediante el uso del stack). No es necesario declarar las variables de usuario de ninguna manera. Estas son definidas en el momento de ser utilizadas por la propia instrucción que las utiliza.

Existe además una variable especial llamada el *contexto*. El contexto no es una variable explícita cuyo contenido pueda ser modificado directamente mediante el uso del comando *SET* (como puede hacerse con el resto de variables). La variable de contexto es una especie de variable global o indicador de situación, cuyo valor se modifica automáticamente por la utilización de ciertos comandos (ver más abajo, en el capítulo 4), es decir, que los comandos, además de realizar su acción, modifican el contexto. También es posible modificar directamente el valor del contexto mediante el comando *SWITCH*. Normalmente, se utiliza el valor del contexto para comprobar el estado del programa y actuar en consecuencia. El capítulo 4 contiene más información acerca de cómo los comandos pueden afectar al contexto.

Acciones

Utilizando R-CODE es también posible hacer que Aibo ejecute acciones predeterminadas. Una acción es un conjunto de movimientos predefinidos identificados por un nombre y que Aibo puede ejecutar. El conjunto de movimientos que compone una acción ha sido desarrollado por los ingenieros de Sony por lo que el programador de R-CODE no tiene que diseñar ninguna acción sino tan solo utilizar las que ya han sido creadas (una lista completa de las acciones disponibles se encuentra en el capítulo 6).

La acción a realizar se especifica mediante el comando *PLAY:ACTION:<la acción a realizar>*. Por ejemplo, para hacer que Aibo se siente, se utilizará la acción *SIT*. El comando necesario para ello será pues *PLAY:ACTION:SIT*.

Algunas acciones requieren el uso de parámetros que deben ser añadidos al comando de ejecución con unos dos puntos adicionales. Por ejemplo, en el caso de que se quiera hacer que Aibo gire sobre sí mismo un cierto número de grados, el parámetro a especificar será el número de grados que se desea que gire. Así para realizar la acción de girar 30º (acción llamada *TURN*), se especificará el comando *PLAY:ACTION:TURN:30*. La lista del capítulo 6 contiene las acciones disponibles con el R-CODE estandar, pero el programador puede crear sus propias acciones mediante el uso de las herramientas RTool y MEdit. El capítulo 12 explica como crear estas acciones e incorporarlas a los

programas del usuario.

Comandos hablados

Mediante el uso de R-CODE es sencillo conseguir que Aibo reconozca comandos hablados, aunque sólo en los idiomas inglés o japonés (según la versión del sistema base instalada en el memory stick). Aibo implementa un sistema de reconocimiento de voz que el programador puede utilizar en sus propios programas. La variable de sistema *AU_ Voice* indica cuando Aibo ha reconocido una palabra, según la lista de palabras que puede reconocer especificada en apéndice C. Así cuando *AU_ Voice* vale 1, Aibo ha reconocido una palabra. El programador puede entonces comprobar el valor de la variable de sistema *AU_ Voice_ ID*, la cual indicará el valor del identificador de la palabra reconocida. El capítulo 10 contiene información más detallada sobre cómo utilizar el reconocimiento de voz.

Tonos y sonidos

De la misma manera que Aibo es capaz de reconocer comandos de voz utilizando R-CODE, es también capaz de reconocer tonos y sonidos producidos por otros Aibos. El reconocimiento de estos sonidos podría ser utilizado por ejemplo para implementar un mecanismo de comunicación entre Aibos, más robusto que la comunicación hablada. Una lista completa de los tonos y sonidos reconocidos se encuentra en el apéndice C, y el capítulo 10 describe cómo implementar dicho reconocimiento.

Chapter 4

Comandos de R-CODE

En este apartado se listan por orden alfabético todos y cada uno de los comandos del lenguage R-CODE. Cada uno incluye una descripción y un ejemplo de utilización. Los símbolos utilizados en la descripción de los mismos es la típica en la descripción de comandos: los símbolos <> indican un parámetro que debe ser introducido. Los símbolos [] indican que el contenido de su interior es opcional.

4.1 Listado de comandos

!

Break. Este comando fuerza una parada del programa en curso. Puede utilizarse de tres formas diferentes

! realiza una parada de emergencia y detiene la ejecución del programa en curso

!! realiza una parada normal y detiene la ejecución del programa en curso

!!! realiza una parada de emergencia, detiene la ejecución del programa en curso y inicializa todo el entorno R-CODE

Ejemplo: detiene el programa en curso si la variable x no vale 87.

```
IF:x:=:87:CALL:prog1
!
:prog1
WAIT
```

:

:<etiqueta>

Indicador de etiqueta. Se utiliza para indicar que el texto de la línea es una etiqueta de salto. Los saltos se producen con los comandos GO, IF o CALL.

La definición de etiquetas tiene un caracter global, es decir, que pueden ser llamadas desde cualquier punto del programa.

Ejemplo: se definen dos etiquetas, prog1 y prog2.

```
:prog1
WAIT
IF:x:=:0:CALL:prog2
WAIT:3
:prog2
```

ADD

ADD:<variable>:<valor>

Suma dos valores enteros. Este comando utiliza dos parámetros: el primero contiene la variable destino del resultado, y el segundo contiene el valor a añadir a la variable destino. El segundo parámetro puede ser un valor fijo o una variable.

Ejemplo: suma 4 más 5

```
SET:x:4
ADD:x:5
GET:x
```

AND

AND:<variable>:<valor>

Realiza el producto lógico entre dos valores, con cada uno de sus bits. Este comando tiene dos parámetros: el primero contiene la variable destino del resultado y el segundo contiene el valor con el que se realizará el producto lógico junto con la variable. El segundo parámetro puede ser una variable.

Ejemplo: producto lógico de 10101010 y 01010101

```
SET:x:0B10101010
AND:x:0B01010101
GET:x
```

ARG

ARG:<variable>

Recupera de la pila un valor suministrado al llamar a una subrutina. Antes de llamar a la subrutina, se coloca en la pila el o los argumentos mediante el uso de la función *PUSH*. Una vez en la subrutina, los argumentos son recuperados mediante el comando *ARG*. Este comando tiene un parámetro y es el nombre de la variable sobre la cual se recupera el valor de la pila. Habrá que llamar tantas veces a la función *ARG* como argumentos se hayan pasado, y en el mismo orden en el que se produjo la inserción en la pila.

Ejemplo: paso de dos parámetros a la subrutina *prog1*

```
...
PUSH:34
PUSH:fx
CALL:prog1
...
:prog1
ARG:arg1  # recupera el valor 34
ARG:arg2  # recupera el valor de fx
...
```

BREAK

BREAK[:<nivel>]

Este comando interrumpe la ejecución de un bucle y sale fuera de él. El parámetro de nivel es opcional y especifical el número de niveles en la estructura de bucles que debe saltarse. Debe ser una constante y el valor por defecto es 1.

Ejemplo: dentro del bucle *FOR,* si detecta una cara sale fuera de él

```
FOR:x:1:10
...
IF:Face:=:1:BREAK
WAIT
NEXT
```

CALL

CALL:<etiqueta>[:<arg>]

Llamada a subrutina. Llama a la subrutina especificada por el parámetro de la etiqueta. El parámetro <arg> especifica el número de parámetros que han sido introducidos en la pila para ser utilizados por la subrutina. El regreso desde la subrutina se realiza por medio del comando *RETURN* o *RET*, que devuelven el punto de ejecución al comando siguiente a la llamada.

Ejemplo:

```
PUSH:45
CALL:prog1:1
...
:prog1
ARG:x
WAIT
...
RETURN
```

CASE

- Formato 1:

CASE:<valor>:<comando>

- Formato 2:

CASE:ELSE:<comando>

Ejecución condicional. Realiza la ejecución del comando especificado según el valor que tenga el *contexto*. El contexto puede contener un valor dependiendo de comandos anteriores. Entonces se ejecutará aquella sentencia *CASE* en la cual el contexto coincida con el parámetro <valor> asociado. Es posible tener varias de estas sentencias seguidas una de otra para captar todos los posible valores de la variable. La última sentencia de un programa así suele ser la sentencia *CASE:ELSE:<comando>* (no es obligatorio) que engloba cualquier otro posible valor de la variable que no se haya tenido en cuenta en las setencias *CASE* anteriores.

Este comando suele combinarse con el comando *SWITCH* el cual se encarga de poner un valor en el contexto.

Ejemplo:

```
SWITCH:x // da al contexto el valor de x
CASE:1:PRINT:"x=1"
CASE:2:PRINT:"x=2"
CASE:3:PRINT:"x=3"
CASE:ELSE:PRINT:"x vale otra cosa"
```

CLR

CLR:SENSORS

Pone a cero todas las variables correspondientes a sensores de detección de eventos, como reconocimiento de cara, pelota, etc.

Ejemplo:

```
CLR:SENSORS
```

CSET

CSET:<var1>:<operador>:<var2>:<valor>

Realiza una comparación de la forma especificada por los tres primeros parámetros <var1> <operador> <var2>. Si la comparación es verdadera, entonces el parámetro <var1>, que debe ser una variable, toma el valor del parámetro <valor>. Es posible colocar varias instrucciones *CSET* una tras otra. En tal caso, cuando el programa encuentra una primera condición verdadera, el resto son ignoradas.

Ejemplo:

```
CSET:x:<:10:1
CSET:x:<:20:2
CSET:x:<:30:3
```

DIV

DIV:<variable>:<valor>

Realiza la división de <variable> entre <valor> y coloca el resultado en <valor>. El parámetro <valor> puede ser un número o una variable. El resultado de la división siempre es un entero, despreciándose la parte decimal del resultado.

Ejemplo: división de 56 entre 3

```
SET:x:56
DIV:x:3
GET:x
```

DO...LOOP

DO[:WHILE|UNTIL:<var1>:<operador>:<var2>]

– comandos –

LOOP[:WHILE|UNTIL:<var1>:<operador>:<var2>]

Ejecuta un bucle de comandos comprendidos entre el comando *DO* y el comando *LOOP*. Si se especifica uno de los parámetros, entonces el bucle se ejecuta mientras se cumpla la condición especificada (*WHILE*) o hasta que se cumpla la condición (*UNTIL*). La condición puede especificarse tanto en el comando *DO* como en el *LOOP*, y está formada por tres parámetros: las dos variables a comparar (<var1> y <var2>) y el operador de comparación (<operador>). Si no se especifica condición *WHILE* ni *UNTIL*, el bucle se ejecuta indefinidamente.

Ejemplo:

```
SET:x:0
DO:WHILE:x:<:10
ADD:x:1
LOOP
```

DUP

DUP

Realiza una copia del primer elemento de la pila y lo coloca en la cola de la misma

Ejemplo: realiza una copia del valor 34 (primer valor de la pila) y lo coloca al fondo de la misma, detrás del número 99.

```
PUSH:34
PUSH:99
DUP
```

EDIT

EDIT
– programa –
END

Carga el programa especificado entre *EDIT* y *END* en la memoria de Aibo, borrando cualquier otro programa que ya estuviese en memoria. Este comando sólo tiene sentido utilizarlo cuando se opera desde la consola inalámbrica.

Ejemplo:

```
EDIT
FOR:x:1:10
PLAY:ACTION:SIT
WAIT
PLAY:ACTION:STAND
WAIT
NEXT
END
```

END

EDIT
– programa –
END

Indica el final de un programa que debe ser cargado en memoria, tecleado a través de la consola. Funciona en conjunción con el comando *EDIT*.

Ejemplo:

```
EDIT
SET:x:0
WHILE:x:<:3
PRINT:"Bucle de espera"
ADD:x:1
WEND
END
```

EQ

EQ

Compara si los dos primeros valores de la pila son iguales, y coloca el resultado de la comparación (TRUE o FALSE) en la pila. Los dos valores comparados son extraidos de la pila por este comando.

Ejemplo:

```
PUSH:x
PUSH:y
EQ
```

```
JF:prog1
...
:prog1
```

EXIT

EXIT

Termina la ejecución de un programa.

Ejemplo:

```
EDIT
SET:x:1000
FOR:i:1:10
DIV:x:i
NEXT
EXIT
END
```

FOR

FOR:<variable>:<desde>:<hasta>[:<paso>]

– comandos –

NEXT

Realiza un bucle de tipo FOR-NEXT donde el parámetro <variable> es el que realiza de contador, desde el valor <desde> hasta el valor <hasta>. Si fuera necesario, es posible introducir el paso de conteo mediante el parámetro <paso>. Es decir, repite los comandos entre *FOR* y *NEXT* mientras incrementa (o decrementa) la <variable> desde <desde> hasta <hasta>, en pasos de <paso>.

Ejemplo:

```
FOR:x:0:10
PRINT:"x=%d":x
NEXT
```

GE

GE

Compara si el segundo valor de la pila es mayor o igual que el primero, y coloca el resultado de la comparación (TRUE o FALSE) en la pila. Los dos valores comparados son extraidos de la pila por este comando.

Ejemplo:

```
PUSH:x
PUSH:y
GE
JF:prog1
```

```
...
:prog1
```

GET

GET:<variable>

Presenta por pantalla (en la consola) el valor de <variable>. Se utiliza con el proposito de debugar programas.

Ejemplo:

```
SET:x:34
GET:x
```

GLOBAL

GLOBAL:<variable>[:<valor inicial>]

Crea una variable de nombre <variable> de ámbito global, en contraposición a una variable local (declarada con *LOCAL*). Esto significa que mantiene su valor cuando es utilizada en subrutinas. Si se especifica <valor inicial>, la variable es inicializada con dicho valor. Si al declarar una variable no se especifica si es global o local, R-CODE genera una variable global por defecto.

Ejemplo:

```
GLOBAL:x:0
```

GO

GO:<etiqueta>

Salta la ejecución del programa a la línea con la etiqueta <etiqueta>. No puede utilizarse para salir de subrutinas, ya que produciría un overflow de la pila, pero sí puede utilizarse para salir de bucles.

Ejemplo:

```
GO:prog1
PRINT:"Esto no se imprime"
:prog1
PRINT:"Esto si se imprime"
```

GT

GT

Compara si el segundo valor de la pila es mayor que el primero, y coloca el resultado de la comparación (TRUE o FALSE) en la pila. Los dos valores comparados son extraidos de la pila por este comando.

Ejemplo: compara dos valores (x e y). Si y no es mayor que x, entonces salta a prog1.

```
PUSH:x
PUSH:y
GT
JF:prog1
...
:prog1
```

HALT

HALT
Detiene el programa en curso y desconecta a Aibo. La desconexión suele durar varios segundos.
Ejemplo:

```
EDIT
PLAY:ACTION:SLEEP
HALT
END
```

IF

- Formato 1

IF:<var1>:<operador>:<var2>:THEN
– comandos_1 –
[ELSE
– comandos_2 –]
ENDIF

- Formato 2

IF:<var1>:<operador>:<var2>:CALL:<etiqueta>[:<argc>]

- Formato 3

IF:<var1>:<operador>:<var2>:BREAK

- Formato 4

IF:<var1>:<operador>:<var2>:<entonces>:<sino>
Realiza una comparación indicada por la secuencia <var1> <operador> <var2>. Si la comparación es verdadera, entonces se ejecuta una de las siguientes acciones, según el formato aplicado:
Formato 1: si la expresión es verdadera, se ejecutan los comandos especificados en comandos_1. En caso contrario se ejecutan los comandos especificados en comandos_2. El bloque *ELSE* y comandos_2 es opcional, pero el comando *ENDIF* es obligatorio para este formato.

Formato 2: si la expresión evaluada es verdadera, se llama a la subrutina especificada como parámetro en el comando *CALL*.

Formato 3: si la expresión es verdadera, se interrumpe la ejecución de un bucle en el que estuviera incluida esta sentencia.

Formato 4: si la expresión es verdadera, entonces salta a la etiqueta <entonces>. En caso contrario salta a la etiqueta <sino>.

Ejemplo:

```
IF:x:=:0:THEN
PRINT:"x vale 0"
ELSE
PRINT:"x no vale 0"
ENDIF
```

INIT

INIT:<nivel>

Inicializa R-CODE segun diversos niveles de inicialización. En el nivel 0 se inicializa completamente R-CODE y es el tipo de inicialización efectuada durante el encendido. Es sólo para uso del sistema y no debe utilizarse en programas del usuario. En el nivel 1 se entra en modo de debugado y no es utilizable actualmente. En el nivel 2 se inicializan el diccionario y la tabla de variables con los valores de inicio. El nivel 3 es que se obtiene durante la conexión con la red wireless. El sistema entra automáticamente en este nivel durante el inicio de la red y no debe usarse en programas de usuario. En el nivel 4 se produce una notificación de la desconexión de la red wireless. Por último, en el nivel 9 se produce un reinicio del robot.

Ejemplo:

```
INIT:2
```

IOR

IOR:<variable>:<valor>

Realiza la suma lógica bit a bit de la <variable> con el <valor>. Este último puede ser otra variable o un valor fijo.

Ejemplo:

```
SET:x:0b01010101
IOR:x:ob10101010
```

JF

JF:<etiqueta>

Extrae un valor de la pila (*POP*) y si es el valor FALSE, entonces salta a <etiqueta>. Suele ser utilizado en convinación con comandos como *GE*,*GT*,*LE*,etc.

Ejemplo: compara dos valores (x e y). Si y no es mayor que x, entonces salta a prog1.

```
PUSH:x
PUSH:y
GT
JF:prog1
...
:prog1
```

JT

JT:<etiqueta>

Extrae un valor de la pila (*POP*) y si es el valor TRUE, entonces salta a <etiqueta>. Suele ser utilizado en convinación con comandos como *GE*,*GT*,*LE*,etc.

Ejemplo: compara dos valores (x e y). Si y es mayor que x, entonces salta a prog1.

```
PUSH:x
PUSH:y
GT
JT:prog1
...
:prog1
```

LAND

LAND:<variable>:<valor>

Realiza una comparación lógica de tipo *AND* entre <variable> y <valor>, colocando el resultado de la comparación en <variable>. Es decir, <variable> tomará el valor TRUE sólo cuando <variable> valga TRUE y <valor> también. En caso contrario, <variable> pasa a tener el valor FALSE.

Ejemplo:

```
PUSH:x
PUSH:y
PUSH:v
PUSH:z
LE // compara si 'x' es menor o igual que 'y'
POP:a // coloca el resultado de la comparación en 'a'
LE // compara si 'v' es menor o igual que 'z'
POP:b // coloca el resultado en 'b'
LAND:a:b // comprueba si 'a' y 'b' son ambas verdaderas
```

LE

LE

Compara si el segundo valor de la pila es menor o igual que el primero, y coloca el resultado de la comparación (TRUE o FALSE) en la pila. Los dos valores comparados son extraidos de la pila por este comando.

Ejemplo: compara dos valores (x e y). Si y no es menor o igual que x, entonces salta a *prog1*.

```
PUSH:x
PUSH:y
LE
JF:prog1
...
:prog1
```

LET

LET:<variable>:<valor>
Asigna el valor <valor> a la variable <variable>.
Ejemplo:

```
LET:x:1
```

LIOR

LIOR:<variable>:<valor>
Realiza una comparación lógica de tipo *OR* entre <variable> y <valor>, colocando el resultado de la comparación en <variable>. Es decir, <variable> tomará el valor TRUE sólo cuando <variable> o <valor> valgan TRUE. En caso contrario, <variable> pasa a tener el valor FALSE.
Ejemplo:

```
PUSH:x
PUSH:y
PUSH:v
PUSH:z
LE // compara si 'x' es menor o igual que 'y'
POP:a // coloca el resultado de la comparación en 'a'
LE // compara si 'v' es menor o igual que 'z'
POP:b // coloca el resultado en 'b'
LIOR:a:b // comprueba si 'a' o 'b' son verdaderas
```

LNOT

LNOT:<variable>:<valor>
Realiza la negación lógica de tipo *NOT* de <valor>, colocando el resultado en <variable>. Es decir, <variable> tomará el valor TRUE cuando <valor> valga FALSE. En caso contrario, <variable> pasa a tener el valor FALSE.
Ejemplo:

```
PUSH:x
PUSH:y
```

```
PUSH:v
PUSH:z
LE // compara si 'x' es menor o igual que 'y'
POP:a // coloca el resultado de la comparación en 'a'
LNOT:b:a // hace la negación de la comparación
```

LOCAL

LOCAL:<variable>[:<valor>]

Define una variable local de nombre <variable>. Si se especifica el parámetro <valor>, entonces la variable es incializada a dicho valor. El espacio que ocupa una variable local es liberado cuando se sale de la subrutina que la ha creado.

Ejemplo:

```
:prog1
LOCAL:x:0
...
RETURN
```

LOOP

DO[:WHILE|UNTIL:<var1>:<operador>:<var2>]

– comandos –

LOOP[:WHILE|UNTIL:<var1>:<operador>:<var2>]

Se utiliza en conjunción con el comando *DO*. Este ejecuta un bucle de comandos comprendidos entre el comando *DO* y el comando *LOOP*. Si se especifica uno de los parámetros, entonces el bucle se ejecuta mientras se cumpla la condición especificada (*WHILE*) o hasta que se cumpla la condición (*UNTIL*). La condición puede especificarse tanto en el comando *DO* como en el *LOOP*, y está formada por tres parámetros: las dos variables a comparar (<var1> y <var2>) y el operador de comparación (<operador>). Si no se especifica condición *WHILE* ni *UNTIL*, el bucle se ejecuta indefinidamente.

Ejemplo:

```
SET:x:0
DO:WHILE:x:<:10
ADD:x:1
LOOP
```

LT

LT

Compara si el segundo valor de la pila es menor que el primero, y coloca el resultado de la comparación (TRUE o FALSE) en la pila. Los dos valores comparados son extraidos de la pila por este comando.

Ejemplo: compara dos valores (x e y). Si y no es menor que x, entonces salta a *prog1*.

```
PUSH:x
PUSH:y
LT
JF:prog1
...
:prog1
```

MOD

MOD:<variable>:<valor>

Realiza la división de <variable> entre <valor> y coloca el resto de dicha división en <variable>. El cociente de la división es descartado. <valor> puede ser tanto un número como una variable.

Ejemplo: obtiene el resto de la división de 9 entre 2.

```
LET:x:9
LET:y:2
MOD:x:y
```

MUL

MUL:<variable>:<valor>

Realiza la multiplicación de <variable> por <valor> y coloca el resultado en <variable>. El parámetro <valor> puede ser un número o una variable. Tener en cuenta que el máximo número permitido es de 31 bits mas signo.

Ejemplo:

```
LET:x:9
LET:y:2
MUL:x:y
```

NE

NE

Compara si el segundo valor de la pila es diferente del primero, y coloca el resultado de la comparación (TRUE o FALSE) en la pila. Los dos valores comparados son extraidos de la pila por este comando.

Ejemplo: compara dos valores (x e y). Si y no diferente de x, entonces salta a *prog1*.

```
PUSH:x
PUSH:y
LT
JF:prog1
...
:prog1
```

NEXT

FOR:<variable>:<desde>:<hasta>[:<paso>]
– comandos –
NEXT

Realiza un bucle de tipo FOR-NEXT donde la <variable> es la que realiza el contador, desde el valor <desde> hasta el valor <hasta>. Si fuera necesario, es posible introducir el paso de conteo mediante el parámetro <paso>. Es decir, repite los comandos entre *FOR* y *NEXT* mientras incrementa (o decrementa) la <variable> desde <desde> hasta <hasta>, en pasos de <paso>.

Ejemplo:

```
FOR:x:0:10
PRINT:"x=%d":x
NEXT
```

NONE

NONE

No realiza operación alguna. El programa continua inmediatamente después con la siguiente línea.

Ejemplo:

```
SET:x:1
NONE
PRINT:"x vale %d":x
```

NOT

NOT:<variable>:<valor>

Realiza la negación lógica bit a bit de <valor> y el resultado lo introduce en <variable>. <valor> puede ser tanto un número como una variable.

Ejemplo:

```
LET:x:0b10101010
NOT:y:x
```

ONCALL (hacer mas experimentos para explicar mejor)

- Formato 1:

ONCALL:<var1>:<operador>:<var2>:<etiqueta>[:<tipo>:<donde>]

- Formato 2:

ONCALL:<-n>

Registra o cancela una rutina de procesamiento de interrupción. Esta instrucción define una rutina que deberá ejecutarse cuando se produzca una condición especificada por <var1>:<operador>:<var2>. Esta condición se escifica

de la misma manera que se hace con el comando IF Cuando se cumpla la condición indicada, el programa saltará automáticamente a la rutina que se haya diseñado a tal efecto. El nombre de la rutina es el que indica el parámetro <etiqueta>.

Los parámetros <tipo> y <donde> son opcionales, pero si uno se especifica, entonces tiene que especificarse también el otro obligatoriamente. Estos indican el tipo de retorno que debe producirse desde la rutina de interrupción.

El segundo formato de este comando sirve para cancelar las <-n> últimas rutinas de interrupción.

Para regresar de la ejecución de la rutina se utiliza el comando *RESUME*.

El capítulo 8 contiene más información acerca del uso de este comando.

Ejemplo: la línea siguiente registra la subrutina *bola_detectada* como el código a ejecutar cuando se produzca la detección de la bola.

```
ONCALL:Pink_Ball:=:1:bola_detectada
```

PLAY

- Formato 1:

PLAY:ACTION:<accion>[:<arg1>:<arg2>:<arg3>]

- Formato 2:

PLAY:MWCID:<numero>

Este comando hace que Aibo ejecute una acción completa, determinada por el parámetro <accion> en el primer formato, o por el parámetro <numero> en el segundo. Una acción es un conjunto de movimientos, luces y sonidos que Aibo ejecuta de principio a fin. Un ejemplo de acción sería el andar. En ese caso, se le podría indicar a Aibo que andase hacia adelante. Otras acciones posibles podrían ser girar hacia un lado, ponerse de pie, golpear la bola, etc... Una descripción más completa de este comando puede encontrarse en el capítulo 5, y una lista completa de las acciones disponibles se encuentra en el apéndice 3.

Los parámetros <arg1>,<arg2> y <arg3> son argumentos que deberán proporcionarse al comando para determinadas acciones especificas. Por ejemplo, para el indicar a Aibo que gire un cierto número de grados, el ángulo de giro sería el parámetro a indicar como argumento. Evidentemente no todos los comandos requieren de argumentos.

El formato 2 de este comando sirve igualmente para especificar una acción a realizar por Aibo, pero mientras que en el primer formato la acción se especificaba mediante una palabra clave (*WALK*, *LIE*, *TURN*,etc...), en el segundo la acción se especifica por un número, el cual corresponde a una acción según una lista de las posibles acciones disponibles (ver apéndice 3). R-CODE incluye varias de estas acciones especificadas por el código MWCID, pero su uso más interesante es en la ejecución de acciones diseñadas por el programador. El capítulo 5 contiene más información acerca de las acciones MWCID, y el capítulo 11

contiene información sobre la creación de acciones nuevas (llamadas *contenidos*) por parte del programador.

Ejemplo:

```
PLAY:ACTION:TURN:45
PLAY:MWCID:26431
```

POP

POP:[<var>]

Extrae un valor de la pila y lo coloca en la variable <var>. Si el parámetro <var> no es especificado, entonces el valor extraído es descartado y eliminado.

Ejemplo:

```
POP:casa
```

PRINT

PRINT:<formato>[:<var1>:<var2>:<var3>:<var4>:<var5>]

Imprime un mensaje en la consola utilizando un formato similar al comando printf() del lenguaje C. Con el parámetro <formato> se especifica qué es lo que se quiere imprimir y con qué formato. El contenido de formato debe ir entre comillas. A continuación, los parámetros <var> indican qué variables se deben imprimir.

Los formatos de las variables disponibles para especificar son %d para indicar que se quiere imprimir una variable en formato decimal, o %x para indicar que se quiere en hexadecimal.

Si no se especifican variables, *PRINT* puede utilizarse para imprimir simples mensajes de texto por la consola.

Ejemplo:

```
PRINT:''La variable X vale %d y la variable Y vale %d'':x:y
```

PUSH

PUSH:<var>

Coloca el valor de la variable <var> en el stack. El parámetro <var> puede ser tanto una variable como un número.

Ejemplo:

```
SET:a:12
PUSH:23
PUSH:a
```

QUIT

QUIT

Realiza una parada de emergencia de cualquier acción que Aibo estuviera realizando y descarta cualquier otra acción que estuviera esperando para ser ejecutada. Notar que este comando se utiliza para detener el movimiento del robot pero no para abandonar la ejecución del programa. También hay que tener en cuenta que si se utiliza este comando cuando Aibo está ejecutando un movimiento, el robot puede quedar parado en una posición indefinida.

Ejemplo: el programa hace caminar a Aibo, pero si detecta que la energía está por debajo del 25% entonces aborta el movimiento.

```
PLAY:ACTION:WALK
IF:Batt_Rest:<:25:THEN
QUIT
ENDIF
WAIT
```

REPEAT

REPEAT

– comandos –

UNTIL:<var1>:<operador>:<var2>

Implementa un bucle de ejecución, en el cual, los comandos especificados entre el *REPEAT* y el *UNTIL* son ejecutados hasta que se cumpla la condición especificada por los parámetros <var1>, <var2> y <operador> (la condición se indica de la misma manera que para el comando *IF*).

Ejemplo:

```
SET:x:0
REPEAT
PRINT:"X vale %d":x
ADD:x:1
UNTIL:x:>:10
```

RESUME

RESUME

Retorna desde una rutina de ejecución de interrupción. La posición dentro del programa a donde retorna depende de los argumentos con los que se haya definido la rutina de interrupción en el comando *ONCALL*. Más información en el capítulo 7.

RET

RET:<contexto>

Retorna de una subrutina y establece como contexto el valor especificado en el parámetro <contexto>. Si el valor de <contexto> es cero, el contexto mantiene el valor que tenía antes de llamar a la subrutina.

Ejemplo:

```
CALL:subrutina_1
CASE:1:PRINT:"Se retorno por RET 1"
CASE:2:PRINT:"Se retorno por RET 2"
CASE:3:PRINT:"Se retorno por RET 3"
...
:subrutina_1
...
RET:1
...
RET:2
...
RET:3
```

RETURN

RETURN[:<valor>]

Retorna desde una subrutina. La subrutina es llamada por el comando *CALL*, y el comando *RETURN* vuelve al punto siguiente en el que se realizó la llamada a subrutina. Si se especifica una valor de retorno (mediante el parámetro <valor>), éste puede ser obtenido tras el retorno de la subrutina mediante la instrucción *POP*.

Ejemplo:

```
CALL:subrutina_1  // llama a la subrutina
POP:valor // aquí ha vuelto de la subrutina y recoge el valor 99
...
:subrutina_1
...
RETURN:99 // finaliza la subrutina y retorna el valor 99
```

RND

RND:<variable>:<desde>:<hasta>

Genera un número aleatorio entre <desde> y <hasta>, y lo coloca en <variable>. Es posible especificar la semilla aleatoria para la generación de números aleatorios mediante el uso del comando *SET:Seed:<semilla>*, donde <semilla> es el valor de la semilla aleatoria.

Ejemplo:

```
RND:x:0:10
```

RUN

RUN

Ejecuta un programa cargado en memoria. Este comando sólo tiene sentido utilizarlo desde la consola telnet. Previamente, se ha debido cargar un programa en la memoria del robot mediante los comandos *EDIT* y *END*.

SET

SET:<variable>:<valor>

Asigna el valor <valor> a la variable <variable>. Funciona igual que el comando *LET* pero incorpora funciones adicionales para la modificación de variables del sistema. Con el comando *SET* es posible modificar el valor de algunas variables de sistema. De hecho, es necesario modificar dichas variables a sus valores iniciales de '0' despues de que hayan sido puestas a '1' por el sistema y se haya producido una detección. Si no se hiciera así, la variable permanecería a '1' todo el rato, incluso despues de haberse terminado la detección del evento. Las variables que pueden modificarse con esta instrucción son:

Head_ON, Head_OFF, Head_Hit, Head_Pat, Head_LONG, Back_ON, Back_OFF, Back_LONG, Jaw_ON, Jaw_OFF, Jaw_LONG, RFLeg_ON, RFLeg_OFF, LFLeg_ON, LFLeg_OFF, RRLeg_ON, RRLeg_OFF, LRLeg_ON, LRLeg_OFF. Una definición más detallada de estas variables puede hayarse en el capítulo 4.

STOP

STOP

Realiza una parada de las acciones que Aibo pudiera estar realizando. Todas las acciones pendientes son descartadas, y si el robot se encuentra en medio de una acción, ésta es finalizada antes de detenerse.

SUB

SUB:<variable>:<valor>

Resta de <variable> la cantidad especificada por <valor>, y coloca el resultado en <variable>. El parámetro valor puede ser un valor concreto o una variable.

Ejemplo:

```
SET:x:12
SUB:x:4
```

SWITCH

SWITCH:<var>

Hace que el *contexto* pase a valer <var>. El parámetro <var> puede ser tanto una variable como una constante. Suele combinarse con el comando *CASE* para implementar condiciones de salto múltiple.

Ejemplo:

```
SWITCH:x
CASE:1:GO:sub1
CASE:2:GO:sub2
CASE:ELSE:PRINT:"Valor de x desconocido"
```

SYNC

SYNC

Al ejecutar este comando durante la ejecución de un programa, éste quedará pausado a la espera de que se le introduzca la cadena SYNC por teclado.

UNTIL

REPEAT

– comandos –

UNTIL:<var1>:<operador>:<var2>

Implementa un bucle de ejecución, en el cual, los comandos especificados entre el *REPEAT* y el *UNTIL* son ejecutados hasta que se cumpla la condición especificada por los parámetros <var1>, <var2> y <operador> (la condición se indica de la misma manera que para el comando *IF*).

Ejemplo:

```
SET:x:0
REPEAT
PRINT:"X vale %d":x
ADD:x:1
UNTIL:x:>:10
```

VDUMP

VDUMP:<variable>

Presenta por la consola el valor de la variable especificada por <variable>, con el formato <nombre de la variable> = <valor de la variable>. Suele utilizarse en tareas de debugado del código.

Ejemplo:

```
VDUMP:x
```

VLOAD

VLOAD:<variable>

Lee el valor de una variable llamada <variable> desde un fichero en el memory stick. En conjunción con el comando *VSAVE* se utiliza para grabar datos en el memory stick y poder recuperarlos después. El valor de la variable es leido del fichero /OPEN-R/APP/PC/AMS/<variable>.SAV

Ejemplo: lee el valor de la variable x desde el fichero /OPEN-R/APP/PC/AMS/x.SAV

```
VLOAD:x
```

VSAVE

VSAVE:<variable>

Graba en el memory stick el valor de la <variable>. En conjunción con el comando *VLOAD* se utiliza para grabar datos en el memory stick y poder recuperarlos después. El valor de la variable es grabado en el fichero /OPEN-R/APP/PC/AMS/<variable>.SAV.

El nombre de la variable a grabar está limitado a 8 caracteres.

Ejemplo: salva el valor de la variable x en el fichero /OPEN-R/APP/PC/AMS/x.SAV

```
VSAVE:x
```

WAIT

- Formato 1:

WAIT

- Formato 2:

WAIT:<ms>

- Formato 3:

WAIT:<parte>

En todos sus formatos, este comando espera a que se finalice una acción para continuar. Se utiliza para sincronizar la ejecución del programa con la ejecución de movimientos del robot. En el primer formato, el programa esperará hasta que se hayan finalizado la acción precedente. En el segundo formato, el programa esperará un número determinado de milisegundos (el rango de milisegundos está entre 1 y 30000, con una resolución de 32 ms). En el tercer formato, el programda detendrá su ejecución hasta que la <parte> especificada haya finalizado su movimiento o ejecución. El parámetro <parte> puede tener uno de los siguientes valores:

MOTION_ALL para esperar a que terminen de moverse todas las partes del robot

MOTION_HEAD para esperar a que termine de moverse la cabeza

MOTION_LEG para esperar a que terminen de moverse las piernas

MOTION_TAIL para esperar a que termine de moverse la cola

MOTION_MOUTH para esperar a que termine de moverse la boca

MOTION_EAR para esperar a que terminen de moverse las orejas

SOUND_ALL para esperar a que terminen de ejecutarse los sonidos

SOUND_SPK para esperar a que termine de ejecutarse sonidos procesados por SP.BIN

LED_ALL para esperar a que terminen de encenderse los LEDs

LED_FACE para esperar a que terminen de encenderse los LEDs de la cara

LED_MODE para esperar a que terminen de encenderse los LEDs de indicación de modo (en el lomo)

Ejemplo:

```
WAIT
WAIT:300
WAIT:LED_FACE
```

WEND

WHILE:<var1>:<operador>:<var2>

— comandos —

WEND

Implementa un bucle de ejecución, en el cual, los comandos especificados entre el *WHILE* y el *WEND* son ejecutados mientras que la condición especificada por los parámetros <var1>, <var2> y <operador> se cumpla. La condición se indica de la misma manera que para el comando *IF*.

Ejemplo:

```
SET:x:0
WHILE:x:<:10
PRINT:"X vale %d":x
ADD:x:1
WEND
```

WHILE

WHILE:<var1>:<operador>:<var2>

— comandos —

WEND

Implementa un bucle de ejecución, en el cual, los comandos especificados entre el *WHILE* y el *WEND* son ejecutados mientras que la condición especificada por los parámetros <var1>, <var2> y <operador> se cumpla. La condición se indica de la misma manera que para el comando *IF*.

Ejemplo:

```
SET:x:0
WHILE:x:<:10
PRINT:"X vale %d":x
ADD:x:1
WEND
```

XOR

XOR:<variable>:<valor>

Realiza la operación binaria OR-exclusiva entre dos valores, con cada uno de sus bits. Este comando tiene dos parámetros: el primero contiene la variable

destino del resultado y el segundo contiene el valor con el que se realizará la operación junto con la variable. El segundo parámetro puede ser una variable.

Ejemplo: OR-exclusiva entre 10101010 y 01010101

```
SET:x:0B10101010
XOR:x:0B01010101
GET:x
```

4.2 Operadores relacionales

A continuación se detalla una lista de los operadores relacionales junto con una descripción de su funcionamiento. Estos operadores pueden utilizarse en la realización de comparaciones o chequeo de condiciones (en comandos como *IF* o *WHILE*).

=

Comparación de igualdad.

==

Comparación de igualdad

<>

Comparación de desigualdad

!=

Comparación de desigualdad

<

Comparación de menor que

<=

Comparación de menor o igual que

>

Comparación de mayor que

>=

Comparación de mayor o igual que

&

Realiza la multiplicación lógica (AND) bit a bit entre dos valores, siendo el resultado FALSE (valor cero) o TRUE (cualquier otro valor)

|

Realiza la suma lógica (OR) bit a bit entre dos valores, siendo el resultado FALSE (valor cero) o TRUE (cualquier otro valor)

^

Realiza la suma lógica exclusiva (XOR) bit a bit entre dos valores, siendo el resultado FALSE (valor cero) o TRUE (cualquier otro valor)

&&

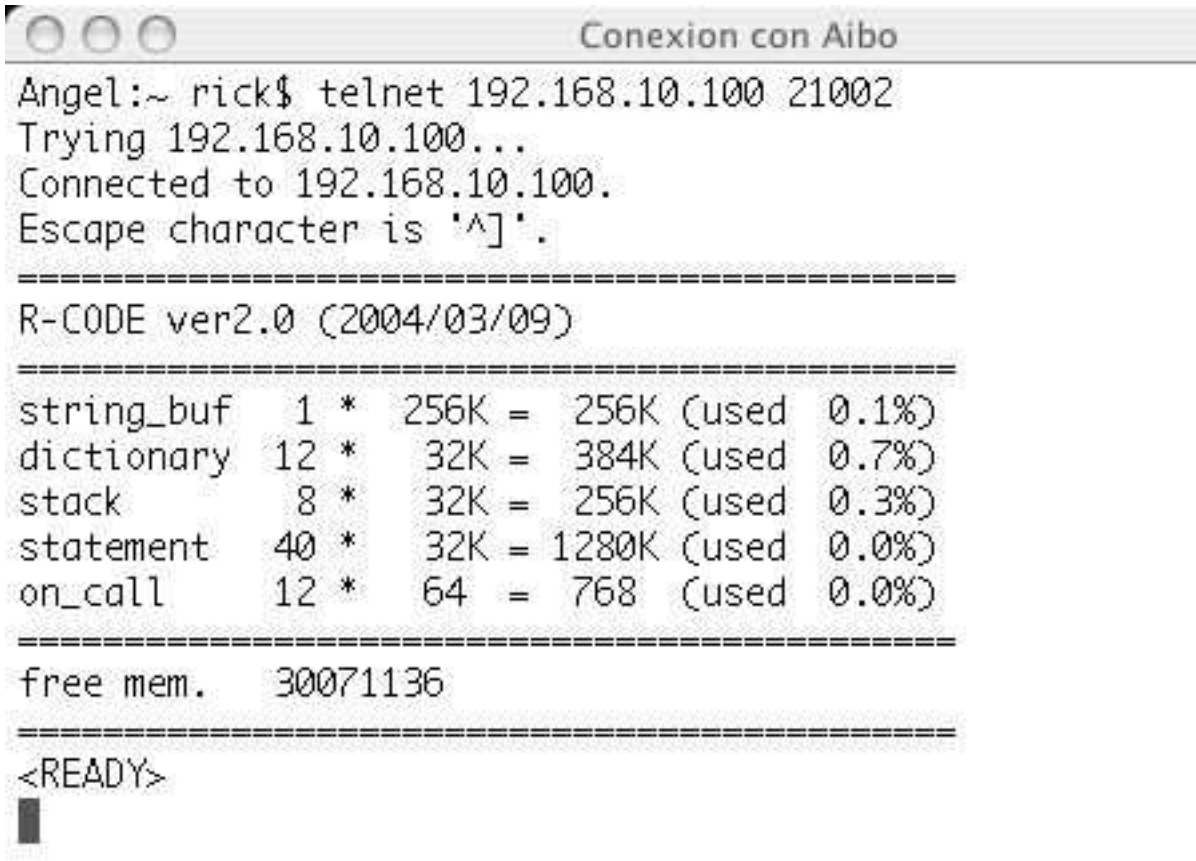
Los operandos son tratados como valores boleanos (FALSE, valor cero, TRUE, cualquier otro valor). Este operador realiza la operación AND entre ambos, siendo el resultado TRUE o FALSE.

||

Los operandos son tratados como valores boleanos (FALSE, valor cero, TRUE, cualquier otro valor). Este operador realiza la operación OR entre ambos, siendo el resultado TRUE o FALSE.

4.3 Debugado de programas

Es difícil encontrar un error en un programa de R-CODE si éste está instalado en el memory stick y se ejecuta desde él en Aibo, ya que el robot no puede indicarnos por qué falla el programa. Por ello, cuando se está desarrollando un programa de R-CODE, suele utilizarse la consola inalámbrica para poder depurar los errores mientras se ejecuta el programa. El procedimiento de debugado consiste en abrir la consola inalámbrica (ver capítulo 3), teclear el programa R-CODE en un editor de textos cualquiera, y una vez finalizado, copiar y pegar el código en la consola inalámbrica, añadiendo los comandos *EDIT* y *END* al principio y final del pegado respectivamente. Una vez el programa se encuentra en la consola, éste se encuentra en la memoria del robot, y puede ejecutarse tecleando el comando *RUN*.



La consola es utilizada entonces para que el programa que se está testeando envíe mensajes de debugado, que permitan al programador comprobar que el programa se comporta correctamente o no, y poder depurar de esta manera posibles errores. En la tarea del debugado se suelen utilizar dos comandos de R-CODE preparados especialmente para enviar mensajes a la consola: *VDUMP* y *PRINT*. Con *VDUMP* se presenta por pantalla el valor de una variable junto con su nombre, mientras que con *PRINT* es posible incluir mensajes mucho más complejos que incluyan el texto que uno quiera junto con la impresión de valores de variables. Consulte el capítulo 4 para más información sobre estos comandos.

Chapter 5

Variables de sistema

El entorno de programación R-CODE proporciona un conjunto de variables del sistema que pueden ser chequeadas (todas) y modificadas (solo algunas), con el fin de conocer el estado del robot o prepararlo para realizar alguna acción. A continuación se incluye una lista detallada con sus descripciones. Tener en cuenta que el sistema es sensible a mayúsculas y minúsculas.

AiboId

Número de identificación de Aibo. Vale 0 cuando no está conectado a una la red inalámbrica, o un número entre 0 y 255, cuando está conectado a una red, y que corresponde al byte menos significativo de su dirección IP.

AiboType

Contiene el número de modelo del Aibo. Para el caso del ERS-7 contiene el valor 7.

Year

Contiene el año de la fecha de Aibo. El año es un valor a partir de 2000.

Month

Contiene el número de mes. Es un valor entre 1 y 12.

Day

Contiene el número del día del mes. Es un valor entre 1 y 31.

Hour

Contiene el número de la hora. Es un valor entre 0 y 23.

Min

Contiene los minutos de la hora. Es un valor entre 0 y 59.

Sec

Contiene el número de segundos de la hora. Es un valor entre 0 y 59 con una resolución de 2 segundos.

Dow

Contiene un número representando el día de la semana, siendo 0 para el Domingo, 1 para el Lunes, etc..., hasta 6 para el Sábado.

Seed

Número con la semilla aleatoria. El valor por defecto es 1.

Power

Valor binario con dos posibles valores: 0 para energía APAGADA y 1 para energía ENCENDIDA.

Status

Indica el estado en el que se encuentra Aibo de entre dos posibilidades: 0 para indicar que Aibo está realizando un inicio normal del programa, o 1 para indicar que el programa ha sido iniciado despues de una caída.

Context

Contiene el valor actual del *contexto* (ver capítulo 4 para definición de contexto).

Wait

Contiene el número de acciones que hay en la cola de acciones, pendientes de ejecutarse.

Clock

Se trata de un contador que se incrementa en 1 unidad cada 32 milisegundos.

Brightness

Variable que indica la luminosidad del ambiente. Es un valor entre 0 y 255.

Face

Variable utilizada para detectar caras. Toma el valor 1 cuando Aibo detecta una cara o el valor 0 cuando no la detecta.

Pink_Ball

Variable utilizada para detectar la pelota. Toma el valor 1 cuando Aibo detecta la pelota o el valor 0 cuando no la detecta.

Pink_Ball_H

Si Aibo ha detectado la pelota, esta variable contiene el ángulo horizontal hacia ella, expresado en grados.

Pink_Ball_V

Si Aibo ha detectado la pelota, esta variable contiene el ángulo vertical hacia ella, expresado en grados.

Pink_Ball_D

Si Aibo ha detectado la pelota, esta variable contiene la distancia hasta ella, expresado en milímetros. El punto de origen de la medida es la cámara.

AIBONE

Variable utilizada para detectar el hueso. Toma el valor 1 cuando Aibo detecta el hueso o el valor 0 cuando no lo detecta.

AIBONE_H

Si Aibo ha detectado el hueso (llamado AIBOne), esta variable contiene el ángulo horizontal hacia él, expresado en grados.

AIBONE_V

Si Aibo ha detectado el hueso, esta variable contiene el ángulo vertical hacia él, expresado en grados.

AIBONE_D

Si Aibo ha detectado el hueso, esta variable contiene la distancia hasta él, expresado en milímetros. El punto de origen de la medida es la cámara.

AU_Voice

Variable utilizada para identificar cuando Aibo ha reconocido un comando de voz. Toma el valor 1 cuando detecta un comando o 0 si no ha detectado nada.

AU_Voice_ID

Una vez el robot ha reconocido un comando de voz, esta variable contiene el identificador ID del comando reconocido, según la lista especificada en el apéndice C.

AU_AiboSound

Chapter 6

Acciones

Aibo es capaz de realizar varias acciones de movimiento, luz y sonido prediseñadas por los ingenieros de Sony con tan solo un comando. R-CODE contiene una librería de acciones que están disponibles al programador para ser directamente utilizadas. En esta sección se describe la lista completa de acciones disponibles así como la forma de utilizarlas en programas.

Las acciones de Aibo se activan mediante el comando *PLAY*. El formato del comando, descrito en el capítulo 4, tiene dos versiones según se utilicen acciones definidas por nombre o acciones definidas por código ID (MWCID). En el primer caso, para ejecutar una acción se utiliza el comando *PLAY:ACTION:<nombre de la accion>:<argumentos>*, donde <nombre de acción> puede ser cualquiera de las acciones listadas más abajo. Los argumentos a añadir al comando dependerán de la acción a realizar. Algunas acciones requieren argumentos y otras no. En el segundo caso, para ejecutar la acción se utiliza el comando *PLAY:MWCID:<número ID de la acción>* donde el número ID se obtiene de una lista de posibles acciones suministrada por Sony (puede encontrarse una lista de estas acciones en el apéndice C). La gracia de la versión MWCID del comando *PLAY* es que permite la utilización de acciones nuevas creadas por el programador. Es posible para el programador crear nuevos acciones e incluirlas en sus programas. Una descripción de como hacerlo se encuentra en el capítulo 12.

Al utilizar el comando *PLAY* dentro de un programa, ocurre que el robot empieza a ejecutar la acción ordenada, mientras que el programa continua ejecutando las líneas siguientes al comando *PLAY*. Dado que la ejecución de la acción lleva algún tiempo, el programa podría desincronizarse, al pensar que la acción está acabada cuando en realidad no lo está. Para evitar este problema se utiliza el comando *WAIT*. Este comando detiene la ejecución del programa hasta que Aibo haya finalizado la acción indicada. De esta forma se consigue que cuando el programa continúe su ejecución, éste sabrá en qué estado se encuentra el robot (el final de la acción).

Para comprender mejor la diferencia entre utilizar o no el comando *WAIT*, se muestra el siguiente programa, en el que se le indica a Aibo que camine una

determinada distancia y que llegado al punto, si detecta la bola rosa que salte a otra subrutina:

```
PLAY:ACTION:WALK:0:10
WAIT
IF:Pink_Ball:=:1:CALL:sub1
```

La introducción en este ejemplo del comando *WAIT* hace que el programa espere hasta que se haya terminado la acción de caminar para emepzar a buscar la bola rosa. Si no se incluyese dicho comando, el robot comenzaría a andar tras el comando *PLAY* y, sin haber terminado la acción de caminar, comenzaría a buscar la bola. En este caso, éste no era el comportamiento deseado, ya que lo que aquí interesaba era que el robot buscase la bola una vez hubiese llegado al punto de búsqueda. Pero podría ser que nos interesase detectar la bola mientras se camina. En tal caso no sería necesario el uso del *WAIT*. Como puede observarse en este ejemplo, el uso del *WAIT* depende de la situación que se quiera programar. Por último, indicar que el comando *WAIT* también tiene diversos formatos de sincronización, que pueden consultarse en el capítulo 4.

Cuando se le indica a Aibo que realice varias acciones una tras otra (por ejemplo, tres acciones con tres comandos *PLAY:ACTION* seguidos), éste implementa una *cola de acciones* y las introduce en ella. Las acciones se irán reproduciendo por orden de llegada a la cola, una después de la otra. Es posible no obstante abortar la reproducción de las acciones en cola mediante el uso de los comandos *STOP* y *QUIT*. Con el primer comando, el programa espera a que se termine la acción en curso y descarta las que estuvieran en la cola. Con el segundo comando, el programa detiene inmediatamente la ejecución de cualquier acción en curso y descarta todas las que estuvieran pendientes en la cola.

6.1 Listado de acciones

A continuación se listan las acciones disponibles mediante el comando *PLAY:ACTION*. La lista contiene el nombre de la acción junto con los parámetros necesarios (si los hay), y una descripción del comando.

SIT

Hace que Aibo se siente.

STAND

Hace que Aibo se ponga de pie.

LIE

Hace que Aibo se tumbe.

WALK:<ángulo horizontal de movimiento>:<distancia>

Hace que Aibo ande en una dirección horizontal determinada, una distancia indicada.

STOP_WALK

Hace que Aibo deje de andar

TURN:<ángulo de giro>

Hace que Aibo gire en torno a su centro un ángulo determinado.

KICK:<ángulo horizontal>:<distancia>

Aibo ejecuta un chute con la pata delantera derecha, en la dirección y distancia especificados.

TOUCH:<ángulo horizontal>:<distancia>

Aibo intenta tocar un objeto situado en el <ángulo horizontal> y a una cierta <distancia>

MOVE_HEAD:<ángulo horizontal>:<ángulo vertical>

Mueve la cabeza a la dirección especificada mediante el ángulo horizontal y el ángulo vertical.

TRACK_HEAD:<objeto a detectar>

Hace que Aibo siga con la cabeza un objeto detectado. El objeto detectado puede ser la pelota (PINK_BALL) o el hueso (AIBONE). En la versión 20040501 de R-CODE sólo es posible detectar la pelota.

SEARCH:<objeto a buscar>

Aibo busca el objeto especificado, la pelota (PINK_BALL) o el hueso (AIBONE). Para ello, mueve la cabeza realizando un barrido de todo el espacio disponible frente a él. Una vez encontrado el objeto, la cabeza apuntará en su dirección y las variables indicadoras (Pink_Ball o AIBONE) cambian a su valor activo (de 0 a 1). En la versión 20040501 de R-CODE sólo es posible detectar la pelota.

SEARCH.HEAD.NORMAL:<objeto a buscar>

Aibo busca el objeto especificado, la pelota (PINK_BALL) o el hueso (AIBONE), en la dirección actual de la cabeza. Una vez encontrado el objeto, la cabeza apuntará en su dirección y las variables indicadoras (Pink_Ball o AIBONE) cambian a su valor activo (de 0 a 1). En la versión 20040501 de R-CODE sólo es posible detectar la pelota.

SEARCH.HEAD.SLOW:<objeto a buscar>

Aibo busca lentamente el objeto especificado, la pelota (PINK_BALL) o el hueso (AIBONE), en la dirección actual de la cabeza. Una vez encontrado el objeto, la cabeza apuntará en su dirección y las variables indicadoras (Pink_Ball o AIBONE) cambian a su valor activo (de 0 a 1). En la versión 20040501 de R-CODE sólo es posible detectar la pelota.

SEARCH.HEAD.NORMALCENT:<objeto a buscar>

Aibo mira hacia el frente y empieza a buscar el objeto especificado, la pelota (PINK_BALL) o el hueso (AIBONE), en la dirección actual de la cabeza. Una vez encontrado el objeto, la cabeza apuntará en su dirección y las variables indicadoras (Pink_Ball o AIBONE) cambian a su valor activo (de 0 a 1). En la versión 20040501 de R-CODE sólo es posible detectar la pelota.

SEARCH.HEAD.SLOWCENT:<objeto a buscar>

Aibo mira hacia el frente y empieza a buscar lentamente el objeto especificado, la pelota (PINK_BALL) o el hueso (AIBONE), en la dirección actual de la cabeza. Una vez encontrado el objeto, la cabeza apuntará en su dirección y las variables indicadoras (Pink_Ball o AIBONE) cambian a su valor activo (de 0 a 1). En la versión 20040501 de R-CODE sólo es posible detectar la pelota.

SEARCH.HEAD.LOWCENT:<objeto a buscar>

Aibo mira hacia abajo y empieza a buscar el objeto especificado, que puede ser la pelota (PINK_BALL) o el hueso (AIBONE), en la dirección actual de la cabeza. Una vez encontrado el objeto, la cabeza apuntará en su dirección y las variables indicadoras (Pink_Ball o AIBONE) cambian a su valor activo (de 0 a 1). En la versión 20040501 de R-CODE sólo es posible detectar la pelota.

PALONE.AUTO.EAR

Mueve las dos orejas durante unos segundos.

PALONE.AUTO.EARSTOP

Detiene el movimiento de las orejas.

PALONE.AUTO.TAILV

Mueve la cola arriba y abajo.

PALONE.AUTO.TAILH

Mueve la cola derecha a izquierda.

PALONE.AUTO.TAILROT

Realiza un movimiento rotatorio con la cola.

PALONE.AUTO.TAILD

Pone la cola en posición recogida.

PALONE.AUTO.TAILSTOP

Detiene el movimiento de la cola.

MOVE.HEAD.FAST:<ángulo horizontal>:<ángulo vertical>

Mueve rápidamente la cabeza en la dirección especificada.

MOVE.HEAD.NORMAL:<ángulo horizontal>:<ángulo vertical>

Mueve la cabeza en la dirección especificada.

MOVE.HEAD.SLOW:<ángulo horizontal>:<ángulo vertical>

Mueve la cabeza lentamente en la dirección especificada.

MOVE.TURN.NORMAL

Hace que Aibo gire a su alrededor.

MOVE.TURN.SLOW

Hace que Aibo gire lentamente a su alrededor.

MOVE.MOVE.NORMAL

Chapter 7

Uso de la pila

R-CODE implementa una pila para el paso de valores a subrutinas, almacenamiento y realización de operaciones aritméticas y de salto. Este es un concepto tomado de la programación en lenguage ensamblador y es la máxima diferencia entre el lenguage R-CODE y otros lenguages interpretados como el BASIC. La pila es una lista de elementos colocados en orden de entrada, implementada automáticamente por el propio sistema R-CODE. El programador puede introducir elementos en la lista, sacarlos, y hacer operaciones con ellos, pero siempre con el primer miembro de la pila. Los demás miembros no pueden ser accedidos hasta que se encuentren en el primer lugar de la pila. El tipo de pila que implementa R-CODE es la pila FIFO (First In First Out). Esto significa que el último elemento introducido en la pila será el último en salir. Por ello, será necesario extraer los elementos anteriores a él de la pila si se quiere accede a él.

7.1 Comandos que operan con la pila

Los elementos se introducen en la pila mediante el comando *PUSH*, y se extraen de ella mediante el comando *POP*. Además de estos comandos, es posible utilizar otros para realizar diversas operaciones con los elementos de la pila como comparaciones, copias, saltos condicionados, basados en el estado de la pila, tal y como se describió en el capitulo 4. No obstante, además de los comandos de manipulación de la pila descritos en dicha sección, existen una serie de comandos adicionales de aritmética y lógica que operan exclusivamente con la pila en lugar de operar con variables. El funcionamiento típico de estos comandos consiste en coger el primero o los dos primeros valores que haya en la pila y ejecutar con ellos una operación aritmética o condicional, colocando el resultado de la misma en la pila. De esta manera es posible implementar saltos condicionales basados en el resultado de una operación sin tener que usar variables. A continuación se describe una lista de todos ellos. Casi ninguno de ellos requiere el pase de parámetros, ya que éstos son obtenidos directamente de la pila.

ADD

Extrae el primero y el segundo elemento de la pila y realiza la suma de ambos, colocando el resultado en la cabeza de la pila.

SUB

Extrae los dos primeros elementos de la pila y al primero le resta el segundo, colocando el resultado en la cabeza de la pila.

MUL

Extrae el primero y el segundo elemento de la pila y realiza el producto de ambos, colocando el resultado en la cabeza de la pila.

DIV

Extrae los dos primeros elementos de la pila y divide el primero por el segundo, colocando el resultado en la cabeza de la pila.

MOD

Extrae los dos primeros elementos de la pila y hace el módulo del primero por el segundo, colocando el resultado en la cabeza de la pila.

AND

Extrae los dos primeros elementos de la pila y realiza la operación AND bit a bit entre ambos, colocando el resultado en la cabeza de la pila.

IOR

Extrae los dos primeros elementos de la pila y realiza la operación OR bit a bit entre ambos, colocando el resultado en la cabeza de la pila.

XOR

Extrae los dos primeros elementos de la pila y realiza la operación XOR bit a bit entre ambos, colocando el resultado en la cabeza de la pila.

NOT

Extrae el primer elemento de la pila y realiza la negación de cada uno de sus bits, colocando el resultado en la cabeza de la pila.

LAND

Extrae los dos primeros elementos de la pila y realiza la operación AND lógica (TRUE o FALSE) entre ambos, colocando el resultado en la cabeza de la pila.

LIOR

Extrae los dos primeros elementos de la pila y realiza la operación OR lógica (TRUE o FALSE) entre ambos, colocando el resultado en la cabeza de la pila.

LNOT

Extrae el primer elemento de la pila y realiza su negación lógica (TRUE o FALSE), colocando el resultado en la cabeza de la pila.

EQ

Extrae los dos primeros elementos de la pila y los compara para ver si son iguales. Si lo son, insertará el valor TRUE en la pila. En caso contrario insertará el valor FALSO.

NE

Extrae los dos primeros elementos de la pila y los compara para ver si NO son iguales. Si NO lo son, insertará el valor TRUE en la pila. En caso contrario insertará el valor FALSO.

LT

Extrae los dos primeros elementos de la pila y los compara. Si el primero es menor que el segundo, insertará el valor TRUE en la pila. En caso contrario insertará el valor FALSO.

LE

Extrae los dos primeros elementos de la pila y los compara. Si el primero es menor o igual que el segundo, insertará el valor TRUE en la pila. En caso contrario insertará el valor FALSO.

GT

Extrae los dos primeros elementos de la pila y los compara. Si el primero es mayor que el segundo, insertará el valor TRUE en la pila. En caso contrario insertará el valor FALSO.

GE

Extrae los dos primeros elementos de la pila y los compara. Si el primero es mayor o igual que el segundo, insertará el valor TRUE en la pila. En caso contrario insertará el valor FALSO.

RND

Tiene dos formatos de utilización, y ambos requieren el paso de parámetros. En el primer caso, el formato utilizado es *RND:<desde>:<hasta>*, con el cual se genera un número aleatorio comprendido en el rango <desde> - <hasta>, y lo introduce el la pila. En el segundo formato se utiliza el comando *RND:<hasta>*, el cual genera un número aleatorio comprendido en el rango 0 - <hasta> y lo introduce en la pila.

La pila también se utiliza para pasar parámetros a una subrutina (ver descripción del comando *ARG* en el capitulo 4, y ejemplo de utilización en el capítulo 8). En este caso, el programa debe introducir en el stack los parámetros que quiera pasar a la subrutina (comando *PUSH*), y a continuación llamarla. Una vez dentro de la subrutina, ésta tendrá que recuperar los parámetros mediante el comando *ARG*. El comando *ARG* recupera de la pila los parámetros introducidos por el programa principal. Se deben realizar tantas llamadas al comando como parámetros se hayan introducido, y en el mismo orden en el que se hicieron.

Chapter 8

Subrutinas

Las subrutinas son pequeños programas dentro del programa principal que implementan funciones. Estas pueden ser llamadas desde diversos puntos del programa principal.

El inicio de una subrutina se define mediante una etiqueta (utilizando el comando de dos puntos :<etiqueta>), y su final viene marcado por el comando *RETURN* o *RET*.

Para llamar a una subrutina desde el programa principal, se utiliza el comando *CALL:<subrutina>*, donde el parámetro <subrutina> es el nombre de la subrutina y tiene que coincidir con el nombre de la etiqueta especificado con el comando dos puntos (:).

Ejemplo de estructura de una subrutina:

```
# programa principal
...
CALL:subrutina_1
...
:subrutina_1
# programa de la subrutina_1
...
RETURN
```

R-CODE permite el paso de valores a una subrutina y la recepción de un valor de retorno desde la misma, mediante el uso de la pila. La pila hace de espacio de intercambio para el paso de valores entre el programa principal y la subrutina. Así, para pasar varios parámetros a la subrutina, el programa principal debe colocar esos parámetros en la pila antes de hacer la llamada a la subrutina. Una vez dentro de la subrutina, ésta deberá recuperar los parámetros pasados mediante el uso del comando *ARG*, siguiendo el mismo orden en el que se introdujeron los parámetros en la pila. Cuando se llama a una subrutina, el valor del contexto queda guardado en la pila y se recupera al retornar de la misma.

El retorno de la subrutina puede realizarse mediante el comando *RETURN* o el comando *RET*. *RETURN* permite realizar un retorno simple o un retorno con

parámetro, en el cual, la subrutina devuelve un parámetro al programa principal que la llamó. *RET* en cambio, realiza un retorno devolviendo un parámetro cuyo valor pasará a ser el valor del *contexto* en el programa principal.

Ejemplo 1: paso de valores entre programa principal y subrutina

```
# programa principal
...
PUSH:valor_1
PUSH:valor_2
CALL:subrutina_1:2 # llama a la subrutina e indica que pasa dos parámetros
POP:valor_final # lee el valor 34 devuelto por la subrutina
...
:subrutina_1
# programa de la subrutina_1
ARG:v1 # recupera valor_1 sobre v1
ARG:v2 # recupera valor_2 sobre v2
...
RETURN:34  # devuelve el valor 34 al programa principal
```

Ejemplo 2: llamada a una subrutina que cambia el valor del contexto

```
# programa principal
...
SWITCH:1  # da al contexto el valor 1
# aquí el contexto vale 1
CALL:subrutina_1
# aquí el contexto vale 7
...
:subrutina_1
...
RET:7
```

Tener en cuenta que cuando se llama a una subrutina se crea una nueva pila y que ésta se destruye una vez se sale de ella.

Chapter 9

Interrupciones

Es posible definir rutinas de interrupción utilizando R-CODE. Esto significa que se puede definir una rutina que deberá ejecutarse cada vez que ocurra un evento determinado. Para ello habrá que definir primero la condición que activará la rutina (evento); segundo, indicar cual es la rutina a activar; y tercero, el tipo de retorno de la rutina de interrupción. Todo ello se hace mediante el comando *ONCALL*.

El comando *ONCALL* se utiliza para registrar una rutina de interrupción. Por registrar se entiende el definir en la lista de interrupciones del sistema una rutina a ejecutarse cuando se produzca un determinado evento (como por ejemplo la detección de una cara). Este comando sirve tanto para definir una rutina de interrupción como para eliminarla. Su formato es el siguiente:

```
ONCALL:<val1>:<operador>:<val2>:<etiqueta>[:<tipo>:<donde>]
```

Los parámetros <val1>,<operador> y <val2> definen la condición que debe cumplirse para que la subrutina se active. Esta es una condición en el formato de sentencia *IF* (ver comando *IF*). El parámetro <etiqueta> define la subrutina que deberá ejecutarse cuando se cumpla la condición especificada. Por ejemplo, si se quiere programar el comportamiento de que cada vez que Aibo detecte una cara realice un saludo sonoro, se podría utlizar el siguiente programa:

```
ONCALL:Face:=:1:saludo
# programa principal
...
# fin programa principal
:saludo
PLAY:MWCID:2507 # ejecuta una acción de saludo
SET:Face:0
RESUME
```

En este ejemplo pueden observarse diversas cosas:

En primer lugar, debe quedar claro que el comando *ONCALL* no llama a la subrutina como tal, sino que registra una rutina a llamar y establece la condición de llamada. La subrutina será activada sólo cuando se cumpla la condifción especificada, en cualquier momento dentro del programa principal. En segundo lugar, el código de la subrutina, debe colocar de nuevo el valor de la variable implicada en su activación al valor de *reposo*. Esto es necesario ya que, tal y como se explicó en el capítulo 5, la variables de detección de estados deben colocarse a su estado de reposo una vez han sido utilizadas, ya que R-CODE no lo hace y sería entonces imposible volver a detectar el evento al mantener el valor 1 para siempre. Tercero, la vuelta desde una subrutina se realiza mediante el comando *RESUME* (no se utiliza ni *RET* ni *RETURN*). La posición de retorno en el programa principal dependerá de cómo se hayan definido los parámetros <tipo> y <donde>.

El parámetro <tipo> es un número entero que especifica el tipo de retorno que debe realizarse desde la rutina de interrupción. Sus posibles valores son los siguientes:

0 retorna a la instrucción en la que se generó la interrupción

1 después de retornar a la instrucción en la que se generó la interrupción, salta a <donde>

2 retorna al lugar en el que se definió la rutina de interrupción (donde se encuentra el comando *ONCALL*)

3 después de retornar al lugar de definición de la subrutina (*ONCALL*), salta a <donde>

4 retorna al principio del programa, vaciando el contenido del stack

5 tras retornar al principio del programa, salta a <donde>

Es posible definir subrutinas de interrupción dentro de subrutinas de interrupción, pero éstas dejan de ser válidas una vez se abandona la primera subrutina de interrupción.

Por último, indicar que es posible cancelar en cualquier momento las últimas n subrutinas de interrupción que hayan sido definidas, mediante el uso del comando ONCALL:<-n>. Este comando cancela las n últimas subrutinas que se hayan definido. No es posible cancelar cada rutina por separado.

Chapter 10

Reconocimiento del habla, tonos y sonidos

10.1 Reconocimiento del habla

Aibo implementa un sistema de reconocimiento del habla que es accesible a través de R-CODE. Es decir, que se pueden hacer programas R-CODE que utilicen el reconocimiento del habla, aunque solamente para reconocer aquellas palabras que R-CODE tiene implementadas internamente (en inglés o japonés según la versión de R-CODE que se haya instalado en el memory stick). No es posible por lo tanto implementar el reconocimiento de palabras nuevas. El apéndice 3 contiene una lista de las 53 palabras que Aibo puede reconocer junto con sus identificadores.

Para incluir el reconocimiento del habla en un programa R-CODE será necesario jugar con dos variables del sistema: AU_Voice y AU_Voice_ID. Cuando Aibo reconoce una de las palabras que es capaz de reconocer, el valor de la variable AU_Voice pasa a valer 1, y el valor de AU_Voice_ID contiene el identificador de la palabra que Aibo ha reconocido. El apéndice C contiene una lista con las correspondencias entre los identificadores y las palabras que representan.

Ejemplo: el programa espera en un bucle mientras no reconoce ninguna palabra. Cuando reconoce algo, pasa a identificar qué es lo que se ha dicho.

```
:inicio
WHILE:AU_Voice:=:0
  # programa a ejecutar mientras
  # Aibo no detecta ninguna frase
  ...
WEND
# en este punto Aibo ha reconocido alguna palabra
SET:AU_Voice:0 # es necesario ponerla manualmente a cero
SWITCH:AU_Voice_ID
```

```
CASE:47:gana   # la palabra reconocida es 'Tu ganas'
CASE:48:pierde # la palabra reconocida es 'Tu pierdes'
CASE:ELSE:inicio # si no se ha reconocido ninguna de esas palabras
                 # se vuelve al principio
```

10.2 Reconocimiento de tonos y sonidos

Aibo también está preparado para reconocer ciertos tonos y sonidos. Estos tonos y sonidos deben ser generados por otro Aibo y de esta forma, es posible establecer una comunicación entre dos robots Aibo para que *hablen* entre ellos. Aibo puede reconocer 68 tonos y 35 sonidos diferentes. Una lista completa de los tonos y sonidos que Aibo puede reconocer y emitir se encuentra en el apéndice C.

La forma de trabajar con el reconocimiento de tonos y sonidos es similar a la del reconocimiento del habla. En este caso se utilizan las variables de sistema AU_AiboTone y AU_AiboTone_ID para el reconocimiento de tonos, y AU_AiboSound y AU_AiboSound_ID para el reconocimiento de sonidos. Cuando Aibo reconoce un tono, la variable AU_AiboTone pasa a valer 1, y AU_AiboTone_ID contiene el identificador numérico del tono que ha reconocido. De igual manera, cuando Aibo reconoce un sonido, la variable AU_AiboSound pasa a valer 1 y la variable AU_AiboSound_ID contiene el identificador de sonido, según la tabla del apéndice C.

Ejemplo:

```
:inicio
IF:AU_AiboSound:=:1:CALL:inicio_comms
...
:inicio_comms
SET:AU_AiboSound:0
IF:AU_AiboSound_ID:<>:1:inicio
PLAY:MWCID:2567
...
RETURN
```

Chapter 11

Ejemplos

A continuación se describe el funcionamiento de los ejemplos incluidos en el R-CODE SDK. R-CODE sólo incluye dos ejemplos básicos, pero que sirven para comprender cómo funcionan los comandos y cómo debe estructurarse un programa de este estilo.

11.1 El ejemplo Greeting.R

El programa Greeting.R espera a reconocer una de dos posibles palabras (*Aibo* o *Hello*). Cuando alguna de esas palabras es reconocida, el robot ejecuta unos movimientos de respuesta. Si detecta alguna otra palabra muestra su enfado con un sonido de tristeza.

```
:START
CALL:1001 # llama a la subrutina de inicialización de nombre 1001
DO # cuando vuelve de la subrutina inicia un bucle  (DO..LOOP)
  WAIT:1 # espera 1 ms
  IF:AU_Voice:=:1:THEN # si Aibo reconoce alguna palabra entonces...(inicio del IF)
    WAIT:1
    SWITCH:AU_Voice_ID # obtiene el identificador de la palabra reconocida
    CASE:1:CALL:1003 # si ha reconocido Aibo llama la subrutina 1003
    CASE:6:CALL:1005 # si ha reconocido Hello llama la subrutina 1005
    CASE:ELSE:CALL:1007 # en otros casos llama a la subrutina 1007
    CALL:1001 # llama a subrutina de inicialización
  ENDIF # fin del IF
  WAIT:1000 # espera un segundo
LOOP # fin del bucle (DO..LOOP)
# rutina de inicialización
:1001
PLAY:ACTION:STAND # pone Aibo de pie
WAIT # espera a que Aibo finalice la acción de ponerse de pie (¡importante!)
SET:AU_Voice:0 # inicializa la variable de reconocimiento de voz a cero
```

```
RETURN
# Aibo ha reconocido la palabra Aibo
:1003
PLAY:ACTION:WALK:0:100 # hace que Aibo ande unos pasos
WAIT # espera a que Aibo finalice la acción
PLAY:ACTION:MOVE_HEAD:0:-50 # mueve la cabeza
WAIT # espera a que Aibo finalice la acción
PLAY:ACTION:MOVE_HEAD:0:0 # mueve la cabeza al centro
WAIT # espera a que Aibo finalice la acción
RETURN
# Aibo reconoce la palabra Hello
:1005
PLAY:MWCID:2750 # ejecuta un mensaje de alegría
WAIT # espera a que Aibo finalice la acción
RETURN
# Aibo ha reconocido otra palabra y está enfadado
:1007
PLAY:MWCID:2471 # ejecuta una combinación de sonidos y luces para mostrar su descont
WAIT # espera a que Aibo finalice la acción
RETURN
```

11.2 El ejemplo Maze.R

Este ejemplo hace que Aibo escape de un laberinto. El procedimiento que sigue es el siguiente: Aibo se mueve unos pasos. Despues se para y hace un escaneo de objetos y distancias a su alrededor. Si detecta que está enfrente de una pared, da media vuelta. Si por el contrario, detecta alguna via de escape, se mueve en esa dirección. Esta secuencia de pasos se repite indefinidamente.

```
:1000
PLAY:ACTION:STAND # pone a Aibo de pie
WAIT # espera a que Aibo finalice la acción
PLAY:ACTION:CLIFF_DETECT_OFF # desconecta la detección de obstáculos
WAIT # espera a que Aibo finalice la acción
DO # inicia el bucle principal
  PLAY:ACTION:MOVE_HEAD:0:0 # mueve la cabeza hasta el centro
  WAIT # espera a que Aibo finalice la acción
  PLAY:ACTION:WALK:0:10000 # camina 10 metros hacia el frente (ángulo 0)
  FOR:t:1:1000 # inicia un bucle FOR
    IF:Distance:<:300:BREAK # si la distancia al objeto más cercano es
                            # menor de 300 mm (una pared) entonces
                            # sal del bucle
    WAIT:1 # espera 1 ms
  NEXT # final del bucle. Sale al cabo de 1 segundo
  PLAY:ACTION:STOP_WALK # detiene el robot
```

```
  WAIT # espera a que Aibo finalice la acción
  PLAY:ACTION:MOVE_HEAD:90:0 # mueve la cabeza a un lado
  WAIT # espera a que Aibo finalice la acción
  PLAY:ACTION:MOVE_HEAD:-90:0 # mueve la cabeza hacia el otro lado
                              # en este caso no espera a su finalización
                              # para realizar otras tareas !!
# esto implica que Aibo realizará un escaneo de distancias
# desde un lado de la cabeza hasta el otro
  SET:dd:0
  WHILE:Wait:>:0 # inicia el bucle que calculará la distancia mayor y su dirección
    SET:d:Distance # d contiene la distancia mayor detectada
    SET:p:Head_Pan # y p contiene la dirección de esa distancia
    IF:d:>:dd:THEN
      SET:dd:d
      SET:pan:p
    ENDIF
  WEND # fin del bucle de cálculo
  VDUMP:dd # muestra los valores detectados por la consola
  VDUMP:pan
  IF:dd:<:300:THEN # si la máxima distancia es menor de 300 mm, significa
                   # que se encuentra enfrente de una pared, y da la vuelta
    PLAY:ACTION:TURN:180 # da la vuelta
    WAIT # espera a que Aibo finalice la acción
  ENDIF
  PLAY:ACTION:MOVE_HEAD:pan:0 # sino, Aibo a encontrado una salida y mira hacia ella
  PLAY:ACTION:TURN:pan # mueve el cuerpo en esa diección
  WAIT  # espera a que Aibo finalice la acción
LOOP # fin del bucle principal
```

Chapter 12

Ficheros de contenidos

Aunque el sistema básico de R-CODE viene con un gran número de acciones, movimientos, secuencias de LEDs y sonidos, es posible para el programador el crear los suyos propios, para luego incluirlos en sus programas. Por ejemplo, es posible para el programador el crear una secuencia de movimientos de baile para Aibo usando un editor de movimientos, y despues invocar este baile desde el propio programa R-CODE utilizando el comando *PLAY*.

A las secuencias de movimientos, los sonidos y los patrones de LEDs utilizados en la creación de una acción para Aibo se les llama *ficheros de contenido*. Esta sección describe cómo crear estos ficheros y cómo utilizarlos en programas própios.

Los pasos básicos para la creación de contenidos que puedan ser utilizados en programas son los siguientes:

1. Creación de los ficheros de contenido. Dependiendo de lo que se quiera incluir en la acción que se vaya a diseñar, estos ficheros serán ficheros que describan movimientos del Aibo (llamados ficheros MTN, con extensión .mtn), secuencias de LEDs (llamados ficheros LED, con extensión .led), o ficheros de audio (pueden ser ficheros de tipo WAV con extensión .wav o de tipo MIDI, con extensión .mid).

2. Conversión de los ficheros de contenido a ficheros de tipo ODA.

3. Edición del fichero de comandos, llamado MWC.CFG, que incluirá la lista de los nuevos contenidos.

4. Conversión del fichero de comandos a un fichero utilizable por Aibo, llamado ERS-7.MWC

5. Instalación de los ficheros de contenido y de comandos al memory stick

6. Invocación de los nuevos comandos de acción desde un programa de R-CODE.

Para la creación de los ficheros de contenido existen diversas herramientas. Este texto se concentrará en el uso de las herramientas oficiales proporcionadas por Sony, pero incluye indicaciones para el uso de otras herramientas creadas por terceras partes.

12.1 Creando ficheros de contenido

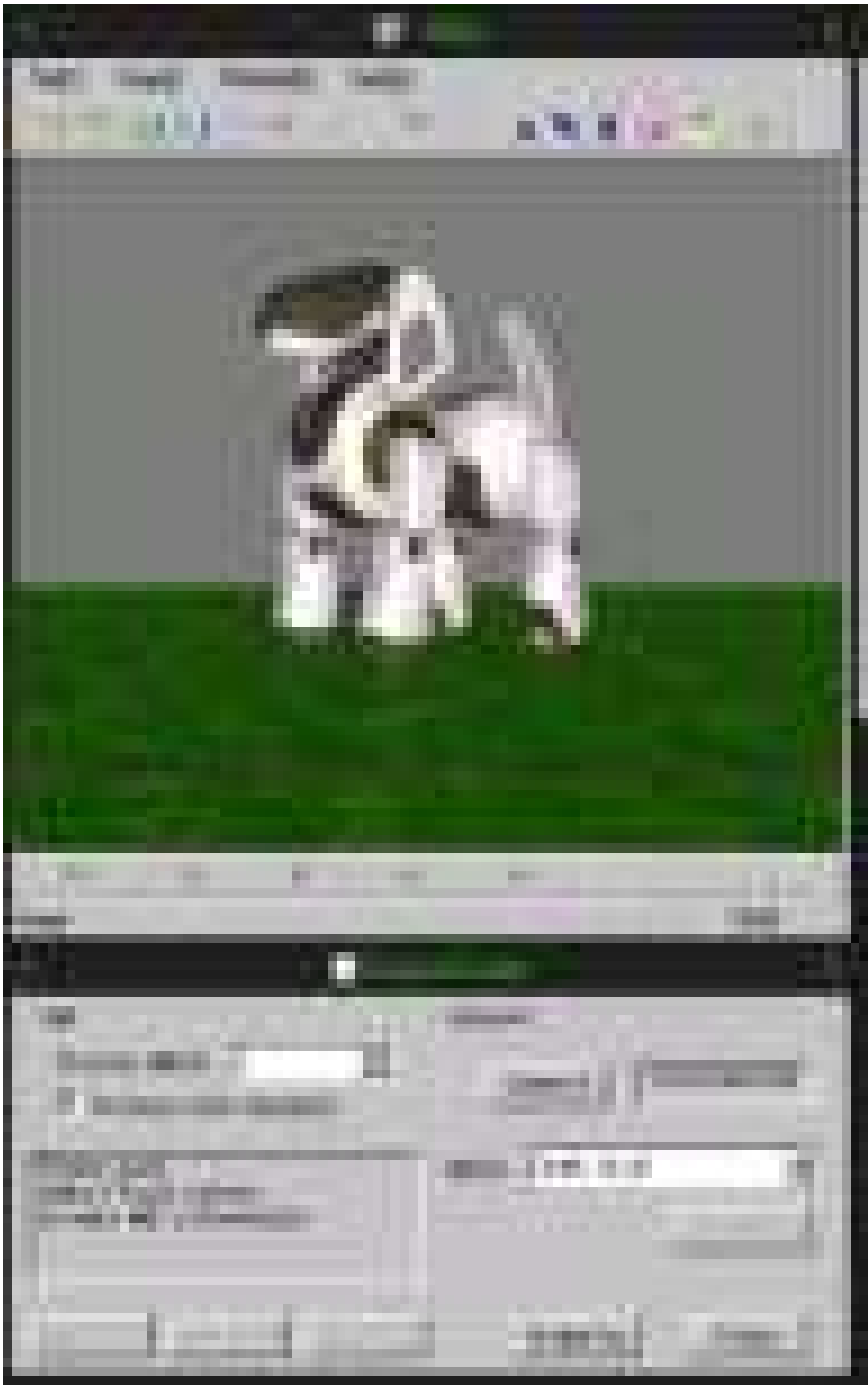
Existen tres tipos diferentes de ficheros de contenido, que deberán ser creados: los ficheros de movimiento, los de audio y los de secuencias LED. No es necesario crear los tres para obtener una nueva acción, ya que puede ocurrir que un movimiento nuevo no requiera de sonido ni de secuencias LED. Los tipos de ficheros de contenidos que serán necesarios para una acción dependerán del diseñador de la misma.

Creando ficheros de movimientos

Los ficheros de movimiento describen secuencias de movimiento de cada una de las partes móviles de Aibo y tienen extensión .mtn. Estas secuencias de movimiento son diseñadas por el programador utilizando un programa llamado *editor de movimientos*. Existen diversos editores de movimientos para Aibo, como por ejemplo Aibo Master Studio, programa comercial de Sony para la edición de movimientos en modelos anteriores al ERS-7; o Skitter, programa de software libre que soporta todos los modelos de Aibo hasta la fecha y que puede ser libremente descargado desde http://www.dogsbodynet.com gratuitamente. No obstante, la utilidad oficial para la creación de movimientos para el ERS-7 es el programa Motion Editor (MEdit) de Sony, que puede ser libremente descargado desde su página web.

Para utilizar el programa MEdit es necesario descargarlo desde la web openr.aibo.com, e instalarlo en un sistema Windows, aunque también es posible utilizarlo es sistemas Linux mediante el uso de Wine[1].

[1] Wine es un emulador de Windows para Linux, el cual permite ejecutar en entornos Linux aplicaciones desarrolladas para Windows. Puede descargarse gratuitamente desde http://www.winehq.com



La instalación y uso del programa es muy simple y viene bien descrito en el manual que le acompaña. No se incluye aquí un manual de uso del programa.

Creando ficheros de audio

Los ficheros de audio contienen sonidos y música que Aibo puede reproducir mientras realiza sus movimientos, y se realizan con dos formatos diferentes, MIDI o WAV, produciendo las extensiones .wav y .mid. Sony no provee de ninguna utilidad especialmente diseñada para la creación de estos ficheros de contenidos, ya que existen multitud de programas libres en la red que permiten crear dichos contenidos. Es pues, tarea del programador escoger un programa u otro que le permita diseñar ficheros WAV y MIDI. Simplemente es enecesario tener en cuenta los siguientes requisitos a la hora de crear dichos ficheros:

1. Los ficheros WAV pueden ser codificados utilizando los siguiente formatos:
   8k 8bits MONO PCM
   16k 16bits MONO PCM
   8k 4 bits MONO IMA ADPCM
   16k 4bits MONO IMA ADPCM
2. Los ficheros MIDI deben codificarse siguiendo el formato SMF Format0.

Es posible utilizar el editor de movimientos Skitter para generar ficheros WAV y MIDI que estén sincronizados con el fichero de movimientos MTN. Skitter incorpora en un solo programa la creación tanto de movimientos como de sonidos WAV y MIDI, por lo que puede ser una buena alternativa para la creación de dichos ficheros de movimiento.

Creando ficheros de LEDs

Los ficheros de LEDs contienen secuencias de encendido y apagado de los diferentes LEDs que dispone Aibo, y se graban con extensión .led. Los ficheros de LED se graban en el mismo formato que los ficheros MIDI, por lo que puede utilizarse el mismo programa que se utilice para diseñar las secuencias MIDI, para las secuencias LED, teniendo en cuenta que los ficheros LED deben grabarse en formato SMF Format0 pero sólo con el track 1 efectivo.

El programa Skitter también contiene un editor de secuencias LED que puede utilizarse para crear los ficheros .led.

12.2 Conversión al formato ODA

Una vez se han creado los ficheros de movimiento, sonido y LEDs (.mtn, .wav, .mid y .led), es necesario convertirlos a los ficheros MOTION.ODA, AUDIO.ODA y LED.ODA. Los pasos a realizar son los siguientes:

1. En primer lugar, es necesario crear un directorio para cada tipo de fichero de contenidos con los siguientes nombres: un directorio MOTION para los ficheros de movimiento, un fichero AUDIO para los ficheros de audio, y un fichero LED para los ficheros LED. A continuación hay que colocar

en cada directorio los ficheros de contenido asociados a cada tipo en su directorio correspondiente, y después ejecutar el programa *RTool.exe* (que forma parte del R-CODE SDK).

2. Una vez ejecutado el fichero *RTool.exe*, seleccionar la pestaña titulada ODA, y colocar las direcciones correspondientes a los directorios de MOTION, AUDIO, LED y *Output Path*. El *Output Path* es el directorio en el que se apareceran los ficheros ODA que se creen.

3. Seleccionar qué ficheros ODA quieren crearse (MOTION, AUDIO y/o LED), y pulsar en *Make ODA* para crearlos.

12.3 Creación del fichero de contenidos MWC

El fichero MWC es el fichero con la lista de los movimientos, sonidos y LEDs nuevos que se hayan creado, y asigna a cada uno de ellos un identificador ID, para que puedan ser utilizados en los programas mediante el comando PLAY:MWCID:<el ID> (el ID asignado en el fichero MWC será el que deberá colocarse en el comando para que Aibo ejecute su nueva acción). Este fichero debe ser creado manualmente y compilado, para ser instalado después en un directorio específico. Esto permitirá el uso de los nuevos contenidos en los programas R-CODE.

El fichero MWC a crear se llama ERS-7.MWC y deberá crearse en dos pasos: uno primero, en el que se creará el fichero en formato ASCII llamado MWC.CFG, y otro segundo de *compilación* del mismo a través de la herramienta *RTool* que dará lugar al fichero final ERS-7.MWC. Un ejemplo del fichero MWC.CFG puede verse a continuación:

```
#MWC 1.0
1 3
26000 3
cmagentMOTIONPERFORMER a_stand#stand_so0r_greet 1 1 0x0 0x0 0
cmagentSOUNDPERFORMER soc_d00greetso0r_x1x 1 1 0x0 0x0 0
cmagentFACELIGHT so12_d00greetso0r_l1f 1 1 0x0 0x0 0
```

Los pasos a seguir para crear el fichero son los siguientes:

1. Creación del fichero MWC.CFG. Se puede crear un fichero nuevo o utilizar uno ya existente para añadirle nuevos comandos. Para crear uno nuevo se realiza lo siguiente:

   (a) Decidir que identificador ID se quiere asignar a cada una de las nuevas acciones que se han diseñado. Este identificador será el que se utilizará en el comando PLAY:MWCID:<ID>. Para acciones nuevas, pueden colocarse identificadores con números comprendidos entre el 26000 y el 27999.

(b) Creación con un editor de texto del fichero MWC.CFG con el siguiente formato:

```
#MWC 1.0
Num1 Num2
Num3 Num4
Texto1 Texto2 Num5 Num6 Num7 Num8 Num9
```

donde Num- y Texto- tienen los siguientes valores:

- Num1: es el número de acciones que incluye el fichero
- Num2: es el número de acciones CMAgent que incluye el fichero
- Num3: es el identificador (ID) que se le asigna a la acción que se va a especificar a continuación en la siguiente línea
- Num4: es el número de acciones CMAgent que se incluyen por cada acción
- Texto1: es el tipo de acción CMAgent. Depende del tipo de contenido que se va a reproducir y los valores posibles son los siguientes:
  - cmagentMOTIONPERFORMER, para una acción en la que se mueva todo el cuerpo.
  - cmagentMOUTHPERFORMER, para unes en las que se mueva la cabeza.
  - cmagentLEGPERFORMER, para acciones en las que se muevan las patas.
  - cmagentTAILPERFORMER, para acciones en las que se mueva la cola.
  - cmagentEARPERFORMER, para acciones en las que se muevan las orejas.
  - cmagentSOUNDPERFORMER, para acciones en las que intervenga el sonido.
  - cmagentMODELIGHT, para acciones relacionadas con los LEDs de modo.
  - cmagentFACELIGHT, para acciones relacionadas con los LEDs de la cara.
  - cmagentEARLIGHT, para acciones relacionadas con los LEDs de la cabeza.
  - cmagentBACKLIGHT, para acciones relacionadas con los LEDs del lomo.
  - cmagentLIVELIGHT, para acciones relacionadas con el LED de red inalámbrica.

- Texto2: es el nombre del fichero de acción CMAgent. Usualmente es el nombre del fichero de contenido sin la extensión.
- Num5: número de veces que la acción CMAgent debe repetirse. Normalmente es 1.
- Num6:
- Num7: parámetro de uso interno. Debe ser siempre 0x0.
- Num8:
- Num9: debe ser siempre 0.

1. Para comenzar la conversión, se ejecutará el programa RTool y seleccionar el fichero de entrada (Input file) como MWC.CFG, el fichero de salida _(Output File) como ERS-7.MWC y el fichero base (Base file) como BASE-7.MWC (que viene con el programa RTool situada en el directorio MWC/).
2. Pulsar el botón Make MWC. El programa generará el fichero binario ERS-7.MWC.

12.4 Instalación en el memory stick

Una vez todos los ficheros han sido creados, habrá que grabarlos en el memory stick, de forma que pueda ser utilizado en programas personales. Para ello, hay que copiar los ficheros MOTION.ODA, LED.ODA y AUDIO.ODA en el directorio OPEN-R/MW/DATA/P/ del memory stick. A continuación, el fichero ERS-7.MWC debe copiarse al directorio OPEN-R/MW/CONF/ del memory stick. Una vez instalados estos ficheros, es posible utilizar los nuevos contenidos y acciones en programas propios utilizando el comando *PLAY:MWCID:<ID>*.

Chapter 13

Apendice A: Instalación del grabador de memory sticks

En este apéndice se tratará la instalación del grabador de memory stick en sistemas Windows y Linux, aunque el trabajo se centrará más en este último sistema. Para iniciar el proceso de instalación será necesario disponer de un grabador de memory sticks con conector USB y un CD con los drivers del fabricante.

Sistemas Windows

La instalación del grabador de memory stick bajo Windows es muy sencilla y no presenta ningún problema. En las últimas versiones del sistema operativo (Windows XP), tan solo conectando el extremo USB en el ordenador, el sistema detectará la conexión y cargará los drivers apropiados sin que el usuario tenga que hacer nada. En versiones anteriores de Windows (Windows 98, Milenium, etc) puede ser necesario instalar manualmente los drivers a partir del CD que contiene los mismos. Para ello, introduzca el CD suministrador con el grabador y siga las instrucciones que irán apareciendo por pantalla.

Sistemas Linux

La instalación del grabador bajo Linux es un poco más complicada y requiere varios pasos. Aunque en la mayoría de distribuciones modernas la detección e instalación del grabador se hace automáticamente cuando se instala la distribución, es posible que sea necesario utilizar el grabador en un sistema Linux ya instalado, para lo cual será necesario seguir los pasos que se describen a continuación.

En primer lugar habrá que compilar el kernel con el soporte adecuado[1]. Asegúrese de que su kernel contiene los siguientes módulos:

[1]Si no sabe cómo compilar el kernel consulte el documento Kernel-COMO disponible en http://es.tldp.org

- SCSI Support (CONFIG_SCSI, scsi.o)
- SCSI disk support (CONFIG_BLK_DEV_SD, sd.o, sd_mod.o)
- SCSI generic support (CONFIG_CHR_DEV_SG, sg.o)
- Support for USB (CONFIG_USB, usb.o)
- Preliminary USB device file system (CONFIG_USB_DEVICEFS, devices.o)
- USB Mass Storage support (CONFIG_USB_STORAGE, usb-storage.o)
- Soporte USB: dependiendo del tipo de hardware de su equipo, tendrá que elegir entre soporte ECHI HCD (CONFIG_USB_EHCI_HCD, usb-ehci-hcd.o), soporte UHCI (CONFIG_USB_UHCI, usb-uhci.o) o soporte OHCI (CONFIG_USB_OHCI, usb-ohci.o)

Compruebe que su kernel dispone de todos esos módulos (cargados como módulos o integrados en el kernel), y si le falta alguno, añádalo y recompile el kernel. Conecte ahora el dispositivo USB al ordenador. Éste debería ser detectado correctamente por el ordenador. Para comprobar si su dispositivo USB ha sido reconocido correctamente teclee:

```
~# cat /proc/scsi/scsi
Attached devices:
...
Host: scsi1  Channel: 00  Id: 00  Lun: 00
  Vendor: Sony   Model: MSAC-US1    Rev: 1.00
  Type: Direct-Access            ANSI SCSI revision: 02
```

Si el sistema funciona correctamente debería obtener una salida parecida a la anterior (dependiendo de su hardware) indicando que el lector USB ha sido identificado por el kernel.

El siguiente paso consiste en identificar el dispositivo de bloques /dev que el kernel ha asignado al grabador USB. Para ello, es posible utilizar el paquete sg3-utils el cual sirve para controlar diversos parámetros en los dispositivos SCSI. Instale en su sistema el paquete sg3-utils[2] y ejecute el siguiente comando:

```
~# sg_scan -i
...
/dev/sg2: scsi1 channel=0 id=0 lun=0 [em]   type=0
   Sony   MSAC-US1    1.00 [wide=0 sync=0 cmdq=0 sftre=0 pq=0x0]
```

Preste atención al dispositivo que el sistema ha asociado a su grabador (en este caso el dispositivo /dev/sg2), y a continuación teclee:

[2]Disponible en http://packages.qa.debian.org/s/sg3-utils.html

```
~# sg_map
/dev/sg0 /dev/scd0
/dev/sg1 /dev/scd1
/dev/sg2 /dev/sda
```

La respuesta al comando anterior indica que el grabador USB (asociado al dispositivo /dev/sg2) puede ser accedido a través del dispositivo de bloques SCSI /dev/sda (en realidad /dev/sda1 ya que /dev/sda es el nombre genérico). Tenga en cuenta que esta información sólo es válida para la sesión actual y podría cambiar en otras sesiones si se acceden a otros dispositivos USB (como cámaras, llaveros USB, etc), por lo que será necesario ejecutar el comando sg_map cada vez que se cambie de sesión o se acceda a otros dispositivos USB.

Una vez identificado el dispositivo de bloques SCSI asociado al grabador, solo queda montarlo para poder acceder a los ficheros. Para ello, cree primero el directorio en el cual se montará el grabador, por ejemplo, /mnt/MS:

```
~# mkdir /mnt/MS
```

Y a continuación introduzca un memory stick en el grabador USB y montelo en ese directorio:

```
~# mount /dev/sda1 /mnt/MS
```

A partir de entonces usted podrá utilizar el memory stick que haya en el grabador como si de una unidad de disco mas se tratase. Puede utilizar cualquier comando de Linux para copiar, pegar o borrar archivos en el memory stick. Recuerde no obstante desmontar el dispositivo antes de extraer el memory stick del grabador, ya que sino podría ocurrir que el buffer de Linux no hubiese grabado los últimos datos enviados. Para desmontar el dispositivo utilice el comando:

```
~# umount /mnt/MS
```

Es posible automatizar el proceso de montaje y desmontaje mediante el uso de los programas hotplug y supermount.

Chapter 14

Apendice B: Configuración de la red inalámbrica

La red inalámbrica no es necesaria en todos los casos de desarrollo pero indudablemente puede suponer una ventaja significativa a la hora de desarrollar más rápidamente los programas de control y transferirlos directamente al robot sin tener que utilizar el grbador de memory sticks. La red wireless permite asimismo realizar experimentos de control remoto y de monitorización del robot.

Dependiendo de la configuración que se disponga en cualto a tarjetas de red inalámbricas y puntos de acceso se podrá configurar una red de una forma o de otra. En este apéndice se tratarán las dos configuraciones más típicas de red inalámbrica: conexión mediante un punto de acceso (llamado modo *managed*), en la cual existe un punto de acceso que hace de router entre ordenadores conectados a una red de cable y la red inalábrica (Aibo y/u otros ordenadores inalábricos), o conexión punto a punto (llamado modo *ad-hoc*), en la cual no existe red de cable ni punto de acceso y los ordenadores se comunican directamente con el robot por medio de una tarjeta de red inalámbrica. Para otras configuraciones, consulte la información suministrada con el robot Aibo.

Nota: en las configuraciones siguientes se ha tomado que las direcciones inalámbricas estarán situadas en el rango 192.168.10.x. Si la red que usted quiere montar funciona en otro rango, adapte las configuraciones de acuerdo con el.

14.0.1 Comunicación a través de un punto de acceso

En este caso, el ordenador que se va utilizar para desarrollo está conectado con un Punto de Acceso (PA) a través de un cable ethernet. El PA se conecta de manera inalámbrica con el robot Aibo, permitiendo de esta manera que el ordenador también lo haga. Este es el llamado modo *managed*, en el cual el ordenador se conecta con el robot utilizando como puente el PA. Para que esta estructura funcione, será necesario configurar PA, el ordenador y el robot.

Configuración del punto de acceso

Se deberán introducir los siguientes parámetros en el PA. La configuración del PA y el método de introducción de los parámetros dependerá del modelo de PA que se disponga. Normalmente estos aparatos suelen traer un software que permite la configuración o permiten el acceso directo a través de un navegador web. Consulte la documentación de su PA para averiguar cómo configurarlo.

ESSID: AIBONET
WEP key: AIBO2
Wireless channel: cualquier canal entre 1 y 11
IP address: 192.168.10.1
Netmask: 255.255.255.0
Operation mode: router

Configuración del ordenador

En este caso, se dispone de una tarjeta ethernet la cual se conecta por cable con el PA. Habrá que configurar entonces la tarjeta para que pueda establecerse la comunicación. En sistemas Windows, utilice el *Panel de Control -> Conexiones de red* para configurar su tarjeta. En sistemas Linux utilice el comando *ifconfig* para realizar la configuración. En cualquiera de los dos casos introduzca los siguientes datos:

IP address: 192.168.10.2 (o cualquier otra disponible en el mismo rango)
Netmask: 255.255.255.0
Gateway: 192.168.10.1

Configuración del robot Aibo

La configuración de la red inalámbrica en el robot, requiere la modificación de los datos contenidos en un fichero especial del memory stick. Este fichero sólo existirá en el memory stick si se ha instalado un sistema base en él (ver capítulo 1 para más información sobre los diversos sistemas base disponibles). Si no comprende todavía qué es el sistema base, debería leer el capítulo 1 al menos, antes de intentar continuar con la instalación de la red inalámbrica.

El fichero de configuración de la red inalámbrica para Aibo se encuentra en el directorio \OPEN-R\SYSTEM\CONF\ del memory stick. Dependiendo del sistema base que haya instalado , OPEN-R o R-CODE, el nombre del fichero será WLANDFLT.TXT o WLANCONF.TXT respectivamente. Si no encuentra el fichero WLANCONF.TXT en el directorio mencionado (sólo para el sistema base R-CODE), vaya al directorio \OPEN-R\SYSTEM\CONF\SAMPLE y copie el fichero WLANCONF.TXT al directorio \OPEN-R\SYSTEM\CONF\. Ahora debería disponer de un fichero WLANDFLT.TXT (para el entorno OPEN-R) o de un fichero WLANCONF.TXT (para el entorno R-CODE) en el directorio \OPEN-R\SYSTEM\CONF\.

Edite ahora el fichero de configuración e introduzca en él los siguientes parámetros:

HOSTNAME=AIBO

ETHER_IP=192.168.10.100
ETHER_NETMASK=255.255.255.0
IP_GATEWAY=192.168.10.1
ESSID=AIBONET
WEPENABLE=1
WEPKEY=AIBO2
APMODE=2 # este modo indica auto-detección
CHANNEL=3
#DNS_SERVER_1=10.0.1.1
#DNS_SERVER_2=10.0.1.2
#DNS_SERVER_3=10.0.1.3
#DNS_DEFDNAME=example.net

Nota: el modo APMODE de auto-detección implica que Aibo se iniciará en modo *managed* o modo *ad-hoc* dependiendo de si, al iniciarse, detecta la presencia de un AP o no. Si detecta un AP, Aibo se iniciará en modo *managed*. En caso contrario lo hará en modo *ad-hoc*. Esta facilidad sólo esta disponible en modelos ERS7.

14.0.2 Configuración en modo punto a punto

En este caso, el ordenador se comunica directamente con Aibo utilizando una tarjeta de red inalámbrica. Este es el llamado modo *ad-hoc*, y necesita configurar la tarjeta inalámbrica del ordenador y el robot Aibo.

Configuración del ordenador

Para configurar la tarjeta en sistemas Windows utilice el *Panel de Control -> Conexiones de red* . Para sistemas Linux tendrá que utilizar el comando *ifconfig* para configurar los parámetros ethernet, y el comando *iwconfig* para configurar los parámetros própios de la red inalámbrica. En ambos casos introduzca los siguientes datos:

ESSID: AIBONET
WEP key: AIBO2
Wireless Channel: cualquiera entre 1 y 11
IP address: 192.168.10.2 (o cualquiera disponible en el mismo rango)
Netmaks:255.255.255.0
Communication mode: ad-hoc (también llamado peer-to-peer)

Configuración del robot Aibo

La configuración del robot es exactamente igual para el caso *ad-hoc* como para el caso *managed*. Utilice, pues, los parámetros e indicaciones de la sección A.1.3.

Chapter 15

Apéndice C: Lista de identificadores para el reconocimiento de voz, tonos y sonidos

15.1 Listado de comandos de voz reconocidos

A continuación se detalla una lista de los comandos de voz que Aibo es capaz de reconocer utilizando R-CODE, y el identificador ID asociado a dicho comando. Desgraciadamente los comandos sólo están disponibles en inglés.

ID Comando
1 AIBO
2 What's your name?
3 Say hello
4 Shake paw
5 Morning
6 Hello sentences sentences
7 Good night
8 See you
9 How are you?
10 Hey AIBO
11 Thanks
12 Sorry
13 Cheer up
14 Banzai
15 That's right
16 That's wrong
17 Good AIBO

18 Don't do that
19 Let's play!
20 Sing a song
21 Dance
22 Show time
23 Pose for me
24 Clown around
25 Show off
26 Say message
27 Let's be secret
28 Open sesame
29 Happy day
30 Stand up
31 Lie down
32 Sit down
33 Turn right
34 Turn left
35 Go forward
36 Go backward
37 Go ahead
38 Stop
39 Faster
40 Slow down
41 Pink ball
42 Right leg kick
43 Right leg touch
44 Left leg kick
45 Left leg touch
46 Ready set go
47 You won
48 You lost
49 Action one
50 Action two
51 Action three
52 Action four
53 Action five

15.2 Listado de sonidos reconocidos

Esta sección detalla los sonidos que Aibo es capaz de reconocer. El listado incluye el identificador ID del sonido, una descripción de lo que significa, y el nombre del fichero MIDI que reproduce este sonido. Todos los ficheros MIDI se encuentran en el directorio *Sample/AiboSound.*

ID Descripción Fichero_MIDI
1 Comienza la comunicación con otro Aibo AS01_Start_AC.mid

2 Inicia el media link mode nº 1 AS02_Start_ML1.mid
3 Inicia el media link mode nº 2 AS03_Start_ML2.mid
4 Inicia el media link mode nº 3 AS04_Start_ML3.mid
5 Finaliza el media link mode, y vuelve al modo autónomo. AS05_End_ML.mid
6 Sube la pata delantera diciendo hola. AS06_Hello.mid
7 Dice adios con la pata. AS07_Bye.mid
8 Se estira en el suelo como si durmiera. AS08_Sleep.mid
9 Niego ligeramente con la cabeza. AS09_Nod_Short.mid
10 Niega profundamente con la cabeza. AS10_Nod_Long.mid
11 Muestra una gran expectación. AS11_Expect.mid
12 Responde positivamente. AS12_Yes.mid
13 Responde negativamente. AS13_No.mid
14 Responde como cuestionando. AS14_Question.mid
15 Pose de alegría. AS15_Joy.mid
16 Pose de mucha alegría. AS16_Happy.mid
17 Pose reluctante. AS17_Disgust.mid
18 Pose de sorpresa. AS18_Surprise.mid
19 Pose de tristeza. AS19_Sad.mid
20 Pose de alivio. AS20_Relief.mid
21 Quiere algo. AS21_Appeal.mid
22 Realiza diversas acciones. AS22_Action.mid
23 Baile. AS23_Dance.mid
24 Canta nº 1. AS24_Song1.mid
25 Canta nº 2. AS25_Song2.mid
26 Baila su baile favorito. AS26_Fav_Dance.mid
27 Pose de miedo. AS27_Fear.mid
28 Bosteza. AS28_Greet.mid
29 Posa para que le saquen una foto. AS29_Pose.mid
30 Eleva las dos patas delanteras. AS30_Cheer.mid
31 Pose de cautela. AS31_Look.mid
32 Pose de vergüenza. AS32_Abash.mid
33 Actua con estilo. AS33_Charm.mid
34 Dice las características de su modelo. AS34_Type_Check.mid
35 Indica que hay una llamada. AS35_Call.mid

15.3 Listado de tonos reconocidos

A continuación se describe el listado de tonos reconocidos por Aibo. El listado incluye el identificador ID del grupo de tonos, los tres tonos que forman el grupo (en formato americano), y el fichero MIDI que reproduce esos tonos. Los ficheros MIDI se encuentran en *Sample/AiboTone*.

ID Tonos (3tonos) Fichero MIDI
1 C5 D5 E5 AT01.mid
2 C5 E5 D5 AT02.mid
3 E5 C5 D5 AT03.mid

4 E5 D5 C5 AT04.mid
5 G5 C5 D5 AT05.mid
6 G5 D5 C5 AT06.mid
7 C5 G5 D5 AT07.mid
8 C5 D5 G5 AT08.mid
9 D5 E5 C5 AT09.mid
10 D5 C5 E5 AT10.mid
11 C5 Eb5 G5 AT11.mid
12 C5 G5 Eb5 AT12.mid
13 G5 Eb5 C5 AT13.mid
14 C5 E5 F5 AT14.mid
15 C5 F5 E5 AT15.mid
16 F5 C5 E5 AT16.mid
17 F5 E5 C5 AT17.mid
18 C5 F5 G5 AT18.mid
19 C5 G5 F5 AT19.mid
20 G5 C5 F5 AT20.mid
21 G5 F5 C5 AT21.mid
22 F5 C5 G5 AT22.mid
23 C5 D5 F5 AT23.mid
24 Eb5 C5 F5 AT24.mid
25 C5 F5 D5 AT25.mid
26 D5 F5 C5 AT26.mid
27 D5 C5 F5 AT27.mid
28 F5 D5 C5 AT28.mid
29 F5 C5 D5 AT29.mid
30 Eb5 F5 C5 AT30.mid
31 C5 Eb5 F5 AT31.mid
32 C5 F5 Eb5 AT32.mid
33 F5 C5 Eb5 AT33.mid
34 F5 Eb5 C5 AT34.mid
35 C6 D6 E6 AT35.mid
36 C6 E6 D6 AT36.mid
37 E6 C6 D6 AT37.mid
38 E6 D6 E6 AT38.mid
39 G6 C6 D6 AT39.mid
40 G6 D6 C6 AT40.mid
41 C6 G6 D6 AT41.mid
42 C6 D6 G6 AT42.mid
43 D6 E6 C6 AT43.mid
44 D6 C6 E6 AT44.mid
45 C6 Eb6 G6 AT45.mid
46 C6 G6 Eb6 AT46.mid
47 G6 Eb6 C6 AT47.mid
48 C6 E6 F6 AT48.mid
49 C6 F6 E6 AT49.mid

50 F6 C6 E6 AT50.mid
51 F6 E6 C6 AT51.mid
52 C6 F6 G6 AT52.mid
53 C6 G6 F6 AT53.mid
54 G6 C6 F6 AT54.mid
55 G6 F6 C6 AT55.mid
56 F6 C6 G6 AT56.mid
57 C6 D6 F6 AT57.mid
58 Eb6 C6 F6 AT58.mid
59 C6 F6 D6 AT59.mid
60 D6 F6 C6 AT60.mid
61 D6 C6 F6 AT61.mid
62 F6 D6 C6 AT62.mid
63 F6 C6 D6 AT63.mid
64 Eb6 F6 C6 AT64.mid
65 C6 Eb6 F6 AT65.mid
66 C6 F6 Eb6 AT66.mid
67 F6 C6 Eb6 AT67.mid
68 F6 Eb6 C6 AT68.mid