Visual Basic:
Programación de robots LEGO

(versión 1.0 en castellano diciembre de 2001)

Koldo Olaskoaga
(Basado en el manual de David Hanley y Sean Hearnek)

El punto de partida para escribir este manual ha sido “Lego Mindstorms Programming with Visual Basic”[1] escrito por David Hanley y Sean Hearne. Una gran parte de esta guía es traducción de dicho manual, aunque con adaptaciones en contenidos y organización. Otras partes son completamente nuevas, como los apéndices A y C (las instrucciones de montaje están hechas utilizando MLCAD, L3p y PovRay). Al inicio de cada capítulo he tratado de resumir los contenidos del capítulo.

[1] Véase la bibliografía.

Contenido

Contenido

Introducción

Los objetivos básicos de este manual se resumen en dos:

- Ofrecer las herramientas necesarias para aprender a programar un robot LEGO MindStorms utilizando Visual Basic.
- Ofrecer un modo atractivo para comenzar el aprendizaje del lenguaje de programación Visual Basic utilizando pequeños robots.

En los primeros capítulos de este manual el control se hará directamente desde el ordenador utilizando formularios. A continuación editaremos programas para transferirlos al robot y ejecutarlos en él, desarrollando de este modo robots autónomos.

He utilizado la versión 6 de Visual Basic, pero también puede utilizarse la 5. En el anexo F se presentan otros dos modos de editar programas transferibles al robot: Visual Basic for Aplications y BrickCommand.

1 Robotics Invention System

1.1. Contenidos de este capítulo

En este capítulo se da una visión general de lo que es el kit Robotics Invention System, dirigida a aquellas personas que no estén iniciadas en su uso.

1.2. Robots Lego

El kit LEGO MindStorms es un conjunto de elementos para montar robots. Además de ofrecer elementos para montar estructuras y transmisiones, contiene el que será el cerebro de nuestro robot, el ladrillo inteligente RCX.

Figura 1.1
el RCX

El RCX es un microcontrolador. Procesa los datos recibidos por medio de una o varias entradas, y toma decisiones sobre la base de dichos datos, controlando las salidas. El RCX tiene tres entradas y tres salidas. En las entradas se conectan los sensores: sensores de luz, sensores de contacto, sensores de temperatura... Las salidas pueden ser motores o luces. Para conectar los motores y sensores al RCX se utilizan los clásicos cables LEGO, por lo que no será necesaria ninguna herramienta para su uso.

1.3. Programación de robots Lego

Cuando se diseña un robot con un objetivo determinado, hay que dotarlo de una estructura y unos comportamientos adecuados. Las respuestas que dará el robot las definiremos por medio de la programación.

Los robots LEGO MindStorms pueden programarse por medio de lenguajes de programación gráficos: RCXCode (lenguaje suministrado con el kit LEGO MindStorms) y Robolab (lenguaje diseñado específicamente para la educación). Para desarrollar programación

avanzada pueden utilizarse diferentes compiladores: Visual Basic, Visual C++, Delphi, Visual Java++...

Una vez compilado el programa en el ordenador se transmite al robot por medio de la torre de infrarrojos. A partir de ahí, el robot funcionará de modo autónomo.

Figura 1.2
los motores y sensores conectados al RCX

1.4. Spirit.ocx

Cuando se instala el CD-ROM que acompaña a LEGO MindStorms y LEGO Technic CyberMaster, el control ActiveX Spirit.ocx se instala automáticamente en el ordenador. Este control permite controlar el RCX desde diferentes entornos de programación, y es imprescindible para poder programar con Visual Basic.

Figura 1.3
Una vez editados los programas, se transfieren al RCX por medio de infrarrojos

2 Primeros pasos con Visual Basic

2.1. Contenidos de este capítulo

En este capítulo tendremos el primer contacto con Visual Basic. En este capítulo veremos ciertos conceptos básicos de programación en Visual Basic, y no comenzaremos con la programación del ladrillo inteligente RCX hasta el siguiente capítulo.

1. Primer contacto con los conceptos básicos de Visual Basic: proyectos, formularios y cuadro de herramientas.
2. Exploración básica de la ventana de propiedades.
3. Procedimiento general para colocar controles en un formulario y modificar sus propiedades.
4. Programas ejecutables.

2.2. Primeros pasos

Una vez que hayas instalado Visual Basic puedes empezar a desarrollar tus proyectos. Para ello inicia la aplicación siguiendo los siguientes pasos:

- Pulsa *Inicio*.
- Elige *Programas*.
- Busca el grupo de programas *Microsoft Visual Basic 6.0*.
- Haz clic sobre *Visual Basic 6.0*.

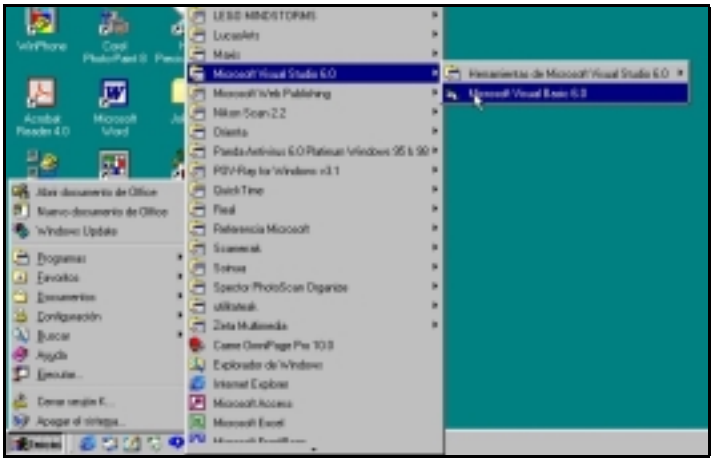


Figura 2.1: busca el icono correspondiente a Visual Basic

Al dar estos pasos se abrirá una ventana de presentación, y a continuación la ventana *Nuevo Proyecto* tal y como se ve en la figura 2.2. Si no aparece dicha ventana, haz clic en el menú *Archivo* y selecciona *Nuevo Proyecto*.

El número de opciones que ofrece la ventana *Nuevo Proyecto* es variable, y depende de la versión de Visual Basic con la que se trabaje.

- Selecciona *EXE estándar* para crear un nuevo proyecto estándar.

Una vez iniciado un nuevo proyecto, el escritorio tomará un aspecto similar al que se puede ver en la figura 2.3.

Aunque todavía no hemos hecho casi nada, hay que guardar el proyecto, y para ello tienes que darle un nombre.

Figura 2.2:
En la ventana de diálogo *Nuevo Proyecto* elige *EXE estándar*

Cuando se guarda un proyecto se crean dos archivos: el archivo de proyecto tiene la extensión .VBP, y se utiliza para guardar la información que Visual Basic necesita del proyecto; el archivo correspondiente al formulario tiene la extensión .FRM, y guarda la información correspondiente al formulario.

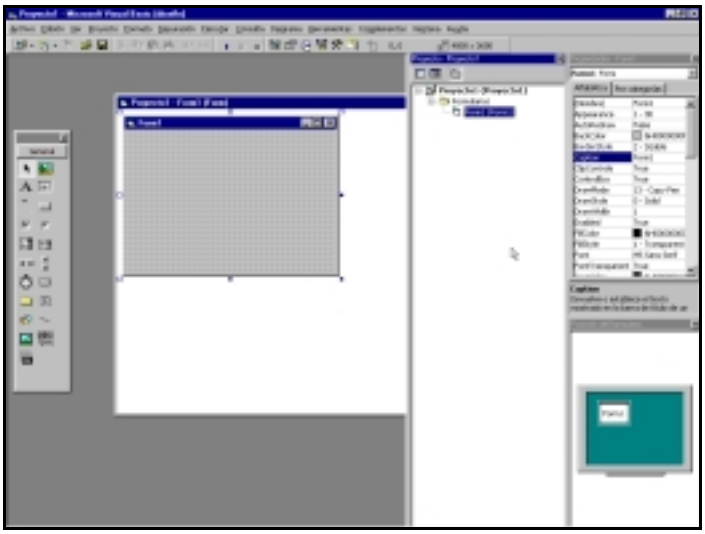


Figura 2.3:
Aspecto de Visual Basic tras iniciar un nuevo proyecto

Antes de guardar un nuevo proyecto conviene crear una nueva carpeta para guardar todos los archivos correspondientes al proyecto, por lo que habrá que seguir los siguientes pasos:

- Selecciona *Guardar Form Como* en el menú *Archivo*. De este modo se guardará el formulario activo.
- Por medio del cuadro de diálogo *Guardar archivo como* puedes elegir el destino en el que quieres guardar el formulario. Los proyectos que vayas creando durante este curso puedes guardarlos en el directorio C:\VBLego\.

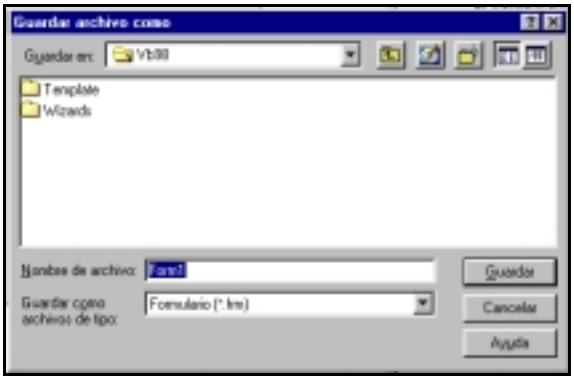


Figura 2.4
Cuadro de diálogo *Guardar como*. Haz clic sobre el botón  para crear un nuevo directorio.

- Selecciona *Crear nueva carpeta* (figura 2.4).
- Escribe el nombre del nuevo directorio (**cap02**) y a continuación pulsa la tecla *Return*.
- A continuación abre la carpeta **cap2** haciendo doble clic sobre ella.
- En el cuadro *Nombre de archivo*, escribe **Hola** (Visual Basic añadirá la extensión .FRM al nombre del archivo al guardarlo).
- Pulsa el botón *Guardar* para guardar el archivo del formulario.
- Selecciona *Guardar proyecto* del menú *Archivo*. Esta opción permite guardar el proyecto actual al completo.
- En el cuadro *Nombre de archivo* escribe **Hola**.
- Pulsa el botón *Guardar* para guardar el archivo del proyecto.

Ahora que has nombrado tanto el formulario como el proyecto, puedes guardar los cambios simplemente seleccionando *Guardar proyecto* del menú *Archivo*, y lo hará utilizando el mismo nombre que anteriormente le hemos dado. Puedes hacerlo también utilizando el icono *Guardar* de la barra de herramientas.

2.3. Ventana de exploración de proyectos

Hemos creado el proyecto Hola.vpb, y está formado por un solo elemento, el formulario hola.frm. En cambio, en otras aplicaciones necesitaremos más elementos. La ventana *Explorador de proyectos* muestra los nombres de todos los archivos que forman el proyecto. Si la ventana *Explorador de proyectos* no está a la vista, selecciónalo en el menú *Ver*.

Los dos iconos de la parte superior sirven para alternar entre las vistas *Objeto* y *Código* (figura 2.5).

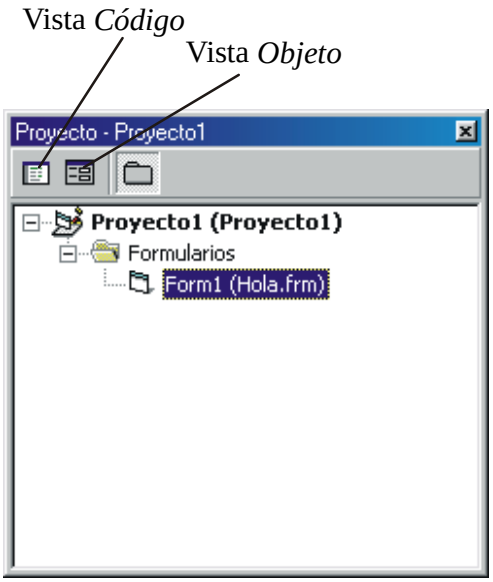


Figura 2.5
Ventana de exploración de proyectos

2.4. Ventana Cuadro de herramientas

En la parte izquierda de tu pantalla puedes ver el cuadro de herramientas, en el que se recogen los controles estándar de Windows. Estos controles, que aparecen en la mayoría de los programas Windows, pueden variar dependiendo de la versión y edición de Visual Basic instalada. Puedes ver el cuadro de herramientas en la figura 2.6. Si no está visible, selecciona *Cuadro de herramientas* en el menú *Ver*.

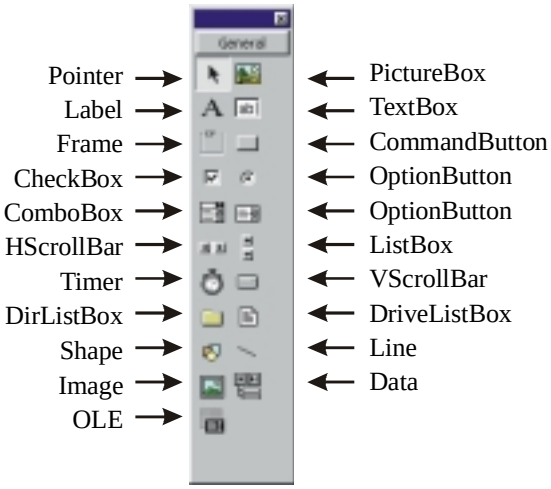


Figura 2.6
cuadro de herramientas Visual Basic

2.5. Controles en el formulario

Ahora vamos a poner un control *CommandButton* en el formulario. Para ello hay que seguir los siguientes pasos:

- Haz un doble clic sobre el icono *CommandButton*. Ello hará que el formulario adquiera la apariencia de la figura 2.7.
- Con el botón seleccionado (aparece rodeado por cuadritos azules), pon el cursor sobre el botón de comando y pulsa y sujeta el botón izquierdo del botón. Mientras

mantienes el botón del ratón presionado, mueve el ratón hacia la parte superior del formulario. El botón se mueve ahora con el ratón. Cuando el botón esté en la posición deseada, soltar el botón izquierdo del ratón.

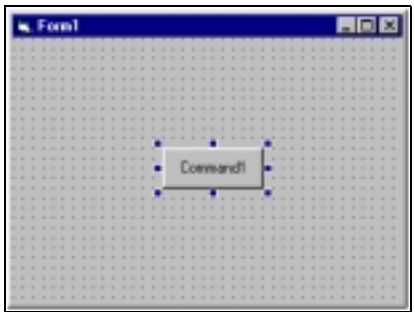


Figura 2.7:
el formulario con el primer botón

2.6. Ventana Propiedades

La ventana propiedades se utiliza para establecer las propiedades de los objetos del proyecto. Si la ventana Propiedades no está a la vista, se selecciona *Ventana Propiedades* del menú *Ver* de Visual Basic.

Las propiedades de un objeto definen cómo se ve y se comporta el objeto. Por ejemplo, un formulario es un objeto. La propiedad *Caption* del formulario define qué texto aparecerá en el título del formulario. El nombre de la propiedad aparece en la parte izquierda de la lista, mientras que el valor actual de dicha propiedad se visualiza en la de la derecha.

Para dar el título "**Programa Hola Mundo**" al formulario de nuestro proyecto, hay que modificar la propiedad *Caption* del formulario.

Haz clic en cualquier parte del formulario, excepto en el botón de comando. En el título de la ventana *Propiedades* podremos leer ahora Propiedades - Form1 si es que está a la vista, y deberían aparecer unos cuadritos azules alrededor del formulario.

Figura 2.8:
la ventana *Propiedades*, por medio de la cual es posible modificar las propiedades aplicables al elemento seleccionado.

- En la ventana *Propiedades*, haz clic en la celda que contiene la palabra *Caption*.
- Sin seleccionar nada más, escribe el texto **Programa Hola Mundo**.

A partir de ahora el formulario tendrá el aspecto de la figura 2.9.

Figura 2.9:
El programa tendrá ahora un título más significativo.

2.6.1. Propiedad "*Nombre*"

En Visual Basic, todos los objetos han de tener un nombre, que es definido por medio de la propiedad *Nombre*. Si miras en la propiedad *Nombre* en el programa Hola podrás ver que se llama **Form1**. Este es el nombre que Visual Basic asigna por defecto a un formulario cuando es creado, pero este nombre no es muy descriptivo, y se puede modificar para hacerlo más útil.

Para modificar la propiedad *Nombre* de un formulario:

- Asegúrate que el formulario está seleccionado.
- Haz clic en la etiqueta *Alfabética* de la ventana *Propiedades*.
- La primera propiedad hace referencia a la propiedad (*Nombre*). Aparece entre paréntesis para que así aparezca en la primera posición de la lista. Haz clic sobre esta primera celda y escribe **frmHola**.

Acabas de cambiar la propiedad *Nombre* a frmHola. Los tres primeros caracteres se utilizan para facilitar la identificación del tipo de control que es el objeto. Esto no es necesario, pero facilita el trabajo y hace que el código sea más fácil de comprender.

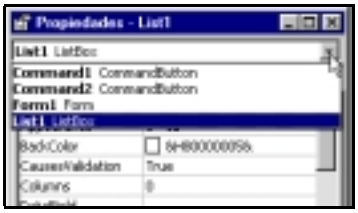


Figura 2.10

Otro modo de moverse por las propiedades de diferentes objetos (en lugar de seleccionar el objeto en el formulario) es por medio de la lista desplegable situada en la parte superior de la ventana *Propiedades*. La ventana *Propiedades* muestra el listado de las propiedades del objeto que aparece seleccionado en la lista desplegable de la parte superior de la ventana. Para ver las propiedades de otro objeto, pulsa el icono flecha hacia debajo de la lista desplegable y selecciona el objeto deseado.

El botón de comando que has creado está dirigido a ser usado para salir del programa, y ahora hemos de cambiar la propiedad Nombre por algo que lo refleje:

- Selecciona la propiedad *Nombre* y cámbiala por **cmdSalir**.

El botón Salir contiene el texto "Command1", el cual es la leyenda por defecto. Para cambiar la leyenda:

- Selecciona la propiedad *Caption* en la lista de propiedades, si es que no está todavía seleccionada, y sustituye el texto por defecto por el texto **&Salir**.

El carácter &, llamado "ampersand", antes de la S en **&Salir**, hace que la S aparezca subrayada en la leyenda del botón. Cuando el programa es ejecutado, pulsar la tecla Alt del teclado al mismo tiempo que la tecla S (Alt + S) tiene el mismo efecto que hacer clic con el botón izquierdo del ratón sobre el botón Salir.

Como te habrás dado cuenta, los nombres de los objetos comienzan por tres letras que describen su tipo. El formulario de este ejercicio es **frmHola** y el botón de comando **cdmSalir**.

Los prefijos para los diferentes tipos de objetos se resumen en la tabla 2.1

Prefijo	Tipo de objeto	Ejemplo
chk	Ckeck box	chkSoloLectura
cbo	Combo box	cboInglés
cmd	Command button	cmdSalir
dlg	Common dialog	dlgAbrirArchivo
frm	Form	frmEntrada
fra	Frame	fraIdioma
gra	Graph	graIngresos
grd	Grid	grdPrecios
hsb	Horizontal scroll bar	hsbVolumen
img	Image	imgIcono
lbl	Label	lblMensajeAyuda
lin	Line	linVertical
lst	List box	lstCódigoPóliza
mnu	Menu	mnuArchivoAbrir
pic	Picture	picVGA
shp	Shape	shpCírculo
txt	Text box	txtÚltimoNombre
tmr	Timer	tmrAlarma
upd	UpDown	updDirección
vsb	Vertical scroll bar	vsbRatio
sld	Slider	sldEscala
tlb	Toolbar	tlbAcciones
sta	StatusBar	staFechaHora

Tabla 2.1

De aquí en adelante siempre que se haga referencia a un objeto se utilizará el valor de la propiedad *Nombre*.

2.6.2. Cambio de la propiedad *Font* del botón cmdSalir

Cuando se crea un nuevo objeto siempre utiliza el mismo tipo de letra en el formulario. Puede ser que interese modificar dicho tipo de letra para adaptar el aspecto del formulario a nuestro gusto. Para ello modificaremos el contenido de la propiedad Font.

Para modificar el tipo de letra del texto del botón **Salir** hay que seguir los siguientes pasos:

- Selecciona el botón **cmdSalir**, y en la ventana *Propiedades* selecciona la propiedad *Font*.

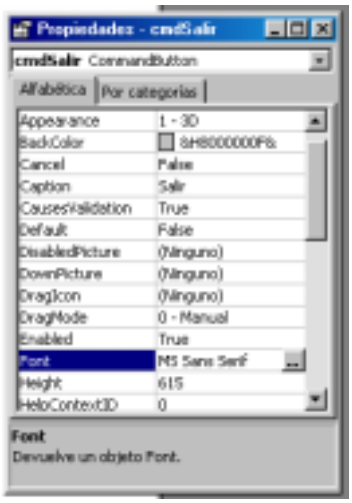


Figura 2.11:
El tipo de letra por defecto para los objetos recién creados es siempre MS Sans Serif. Puedes modificarlo en la ventana *Propiedades*.

Por ahora, el tipo de letra es MS Sans Serif, pero vamos a cambiarlo por Arial:

- Haz clic sobre el icono que contiene tres puntos (puntos suspensivos) a la derecha de la palabra *Font*
- Cambia la fuente y el tamaño a Arial 10, y pulsa a continuación el botón *Aceptar*.

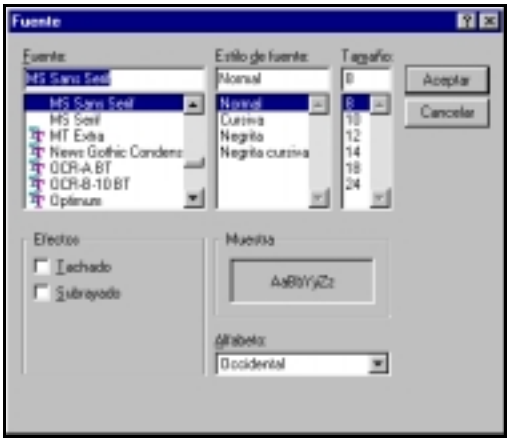


Figura 2.12:
El cuadro de diálogo *Fuente*.

El texto del botón **cmdSalir** ha cambiado su apariencia.

Figura 2.13:
El tipo de letra de la leyenda del botón de comando ha cambiado.

Vamos a añadir ahora más botones al formulario:

- Como antes, haz doble clic sobre el icono *CommandButton* del cuadro de herramientas.
- Arrastra el recién creado botón a la parte izquierda del formulario.

Ahora vas a crear otro botón en el formulario, pero esta vez vas a utilizar un método alternativo.

- Haz un clic sobre el icono *CommandButton* del cuadro de herramientas y mueve el cursor hasta el formulario.
- Mueve el cursor hasta la posición en que deseas que esté uno de los cuatro vértices del botón.
- Pulsa el botón izquierdo del ratón, y, sin soltar el botón del ratón, arrastra el cursor hasta la posición en que deseas que esté el vértice opuesto del botón, donde soltarás el botón del ratón.

Figura 2.14:
El formulario tiene un nuevo botón.

Ahora vamos a modificar el tamaño del botón de comando:

- Haz clic sobre el botón *Command1*. Si lo has hecho correctamente, unos tiradores azules aparecerán alrededor del botón.
- Coloca el cursor sobre el tirador inferior central. El aspecto del cursor deberá tomar la forma de una flecha con dos puntas.
- Pulsa el botón izquierdo del ratón y sin soltarlo arrástralo hacia abajo para agrandar el botón.
- Crea el botón de comando *Command2* utilizando el procedimiento anterior.

Figura 2.15:
Añade un nuevo botón al formulario y modifica el tamaño de los dos.

2.6.3. Cambio de las propiedades de los nuevos botones

Del mismo modo que hemos hecho la orden **cmdSalir**, modificaremos algunas propiedades de los nuevos botones, entre ellas *Nombre* y *Caption*

- Selecciona el botón *Command1*.
- Cambia la propiedad *Nombre* a **cmdHola.**
- Cambia el valor de la propiedad *Caption* a **&Hola Mundo**.
- Cambia la fuente a Arial tamaño 10. Todos los elementos del formulario han de aparecer con este tipo y tamaño de letra.
- Haz lo mismo con el botón *Command2*, dándole el nombre **cmdBorrar**, y cambia la propiedad *Caption* por **&Borrar**

Figura 2.16:
aspecto que deberá tener el formulario tras los cambios anteriores

La leyenda del botón **cmdHola** aparece en dos líneas, si deseas encajar en texto en una sola línea sigue los siguientes pasos:

- Selecciona el botón **cmdHola**.
- Arrastra el tirador central de la derecha para alargar el botón.

Si quieres que los dos nuevos botones tengan el mismo tamaño, o los tres, puedes utilizar la opción que ofrece el menú *Formato*:

- Selecciona todos los botones a los que hay que dar el mismo tamaño. Para ello, ve pulsando sobre cada uno de los botones mientras mantienes pulsada la tecla <**Shift**>.
- En el menú Formato, selecciona *Igualar tamaño* ⇒ *Ambos*. Los botones ahora presentarán el mismo tamaño.

Si quieres alinear los botones horizontalmente, selecciona los botones deseados y selecciona a continuación *Formato* ⇒ *Alinear* ⇒ *Inferior*.

Experimenta con las diferentes opciones del menú *Formato* para conocer lo que ofrecen.

2.6.4. El control Cuadro de Texto (TextBox)

Ahora vamos a añadir un nuevo objeto al formulario, un cuadro de texto. Un cuadro de texto es un cuadro que podemos colocar en el formulario, y que puede utilizarse para introducir código en el programa, o para visualizar procedente de una operación del programa. El elemento *TextBox* es el icono del cuadro de herramientas que tiene las letras *ab* sobre él (). Si colocas el cursor sobre dicho icono, el texto *TextBox* aparecerá dentro de un rectángulo amarillo.

- Haz un clic sobre el icono *TextBox* y mueve el cursor hacia el formulario.

- Mueve el cursor hasta la posición en que deseas que esté uno de los cuatro vértices del objeto *TextBox*, y arrastra el cursor hasta la posición en que deseas que esté el vértice opuesto.

Cuando sueltes el botón del ratón, el cuadro de texto y su contenido por defecto se harán visibles.

Figura 2.17 :
cuadro de texto en su configuración por defecto.

Vamos a cambiar algunas propiedades del cuadro de texto:

- Asegúrate que el cuadro de texto que acabas de crear está seleccionado.
- Cambia la propiedad *Nombre* a **txtHola**.
- Borra el contenido de la propiedad *Text* (actualmente Text1), para que no aparezca nada cuando el programa se ejecute.
- La propiedad *Alignment* (alineación) es por defecto *0-left Justify* (justificado a la izquierda), lo cual quiere decir que el texto se alineará en la parte izquierda del cuadro de texto cuando el programa sea ejecutado. En este programa queremos que el texto aparezca centrado en el cuadro de texto, por lo cual deberemos cambiar esta opción y darle el valor *2-Center*. Para ello, utiliza la lista desplegable que aparece cuando se pulsa la flecha que señala hacia abajo.

También puedes dar a la propiedad *Multiline* el valor True, o Visual Basic ignorará la configuración de la propiedad *Alignment* (si el texto supera la anchura de una línea y la propiedad *Multiline* tiene el valor False, el texto no tomará en cuenta la propiedad *Alignement*).

- Cambia la propiedad *Font* de **txtHola** a Arial tamaño 10.

2.7. Ejecución del programa

Si quieres ver el programa en marcha tal y como está por ahora:

- Guarda tu trabajo seleccionando *Guardar proyecto* del menú *Archivo* (o por medio de un clic sobre el icono *Guardar proyecto* de la barra de herramientas).
- Selecciona *Iniciar* del menú *Ejecutar* (se puede hacer también pulsando la tecla de función F5 o pulsando el botón Iniciar de la barra de herramientas).

Tal y como podrás observar, si pulsas cualquiera de los botones del formulario no pasará nada. Esto es porque todavía no hemos asignado código a estos botones.

- Para salir de la aplicación, pulsa el botón ☒ en la parte superior izquierda de la ventana.

2.8. Agregar código a los objetos

Visual Basic es un lenguaje dirigido a eventos. Cuando un evento es detectado, el proyecto accede al procedimiento de evento adecuado. Los procedimientos de evento se utilizan para comunicar al ordenador qué es lo que debe hacer en respuesta a un evento.

En nuestro programa, un ejemplo de lo que un evento puede ser, es el pulsar el botón cmdSalir. Por ahora, cuando presionamos este botón ocurre un evento, pero todavía no hemos asociado un procedimiento de evento a este evento.

Ahora añadiremos código a este evento:

- Haz un doble clic sobre el botón cmdSalir. Entonces se abre la ventana de código con un shell para tu sub procedimiento. Es decir, la primera y última línea de tu sub procedimiento están ya en su sitio.

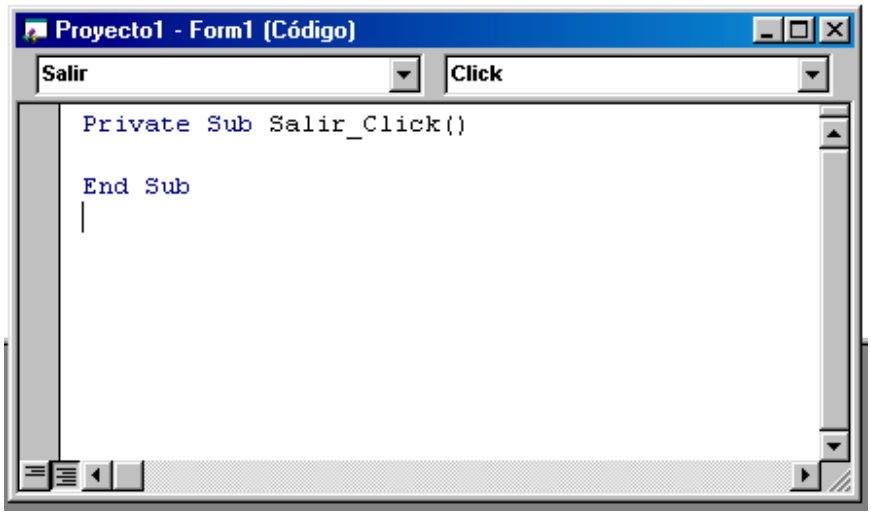


Figura 2.18 la ventana código con la primera y última lista del procedimiento

Tal y como se puede ver en la figura 2.18, la lista desplegable de la parte superior izquierda (la lista de objetos) muestra el nombre del objeto (cmdSalir) y la lista desplegable de la parte superior derecha (la lista de procedimientos) muestra el nombre del evento “Click”.

- Pulsa el tabulador del teclado una vez para sangrar y a continuación escribir la siguiente instrucción:

```
End
```

En la ventana *Código* deberá verse lo siguiente:

```
Private Sub cdmSalir_Click()
    End
End Sub
```

Sangrar las instrucciones de un procedimiento facilita su lectura e interpretación.

- Guarda tu trabajo y ejecuta el programa. Lo puedes hacer pulsando el botón de aspecto similar a la tecla *Play* del video que aparece en la barra de herramientas.
- Pulsa el botón Salir (o las teclas <**Alt + S**> simultáneamente), con lo que el programa finalizará su ejecución.

A continuación, asociaremos procedimientos de evento al resto de los botones.

2.8.1. Añadir código al botón cmdHola

Vamos a empezar añadiendo código al botón *cmdHola*:

- Vuelve a la ventana *Objeto*. Puedes hacerlo seleccionando *Objeto* en el menú *Ver*, o pulsando el icono central de la parte superior de la ventana *Propiedades*.
- Haz un doble clic sobre el botón cdmHola. La ventana de código volverá a aparecer con el shell del procedimiento cdmHola_Clic().
- Escribe lo siguiente:

```
TxtHola.Text = “Hola Mundo”
```

Te habrás dado cuenta que al teclear el punto que va a continuación de txtHola se presenta una lista de opciones. Estas son las únicas opciones que puedes elegir para el elemento actual, en este caso un cuadro de texto. Puedes elegir cualquiera de ellas de tres modos diferentes: utilizando las flechas arriba y abajo, y pulsando la barra espaciadora para acabar; movientote hacia arriba o abajo con el ratón para finalizar con un clic sobre el elemento deseado; o escribiendo tu mismo la palabra.

Esta instrucción asigna el valor “Hola mundo” a la propiedad Text de txtHola. Al ser una cadena de texto, es necesario escribirla entre comillas.

2.8.2. Añadir código al botón cmdBorrar

Ahora vamos añadir código al botón **cmdBorrar**. El procedimiento a seguir es el mismo del botón anterior:

- Vuelve a la ventana *Objeto* otra vez.
- Haz un doble clic sobre el botón *cdmBorrar*. La ventana de código volverá a aparecer con el shell del procedimiento cdmBorrar_Clic().
- Escribe el siguiente código:

```
TxtHola.Text = “”
```

Esta instrucción asigna el valor nulo a la propiedad Text de txtHola. En otras palabras, borra el contenido del cuadro de texto. La ventana de código tendrá la apariencia de la figura 1.19.

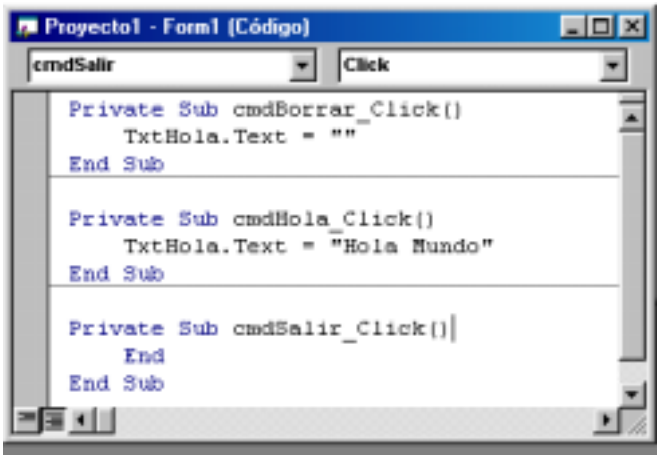


Figura 2.19:
Vista del código actual

2.8.3. Ejecución del programa

Ahora el programa Hola está completo. Para ver el producto final:

- Guarda tu trabajo.
- Ejecuta tu programa.

Figura 2.20:
Cuando ejecutes tus programas, prueba todos los botones para comprobar que funciona correctamente.

- Pulsa el botón *Hola mundo* y las palabras "Hola mundo" aparecerán en el cuadro de texto.
- Pulsa el botón *Borrar* y el contenido del cuadro de texto se borrará.
- Pulsando en el teclado las combinaciones <**Alt + H**> y <**Alt + B**> respectivamente obtendremos el mismo efecto.
- Para finalizar el programa, pulsa el botón *Salir* (o la combinación <**Alt + S**>).

2.9. Creación de un archivo ejecutable

Tal y como está ahora el programa, sólo se puede ejecutar en un entorno Visual Basic. Si deseas que se pueda ejecutar como cualquier otro programa Visual Basic debes hacer lo siguiente:

- Selecciona *Generar Hola.exe* del menú *Archivo*.
- En el cuadro de diálogo que aparece a continuación, el nombre del ejecutable es Hola.exe, aunque si quieres cambiarlo puedes hacerlo.
- El directorio en el que va a ser creado el ejecutable aparece en la parte superior del cuadro de diálogo. Debería ser el mismo que hemos creado anteriormente (cap01). Si es así pulsa el botón *Aceptar*.
- El programa ejecutable se creará en el directorio *cap01*.

A partir de ahora podrás ejecutar la aplicación Hola del mismo modo que cualquier otra aplicación Windows..

2.10. Metodología

El método que hemos utilizado para presentar el código para los programas, ha sido algo desordenado y desorganizado. De aquí en adelante, los datos se presentarán por medio de una detallada tabla que recogerá todos los elementos que es necesario colocar en el formulario, sus nombres y los valores que hay que dar a sus propiedades. No todas las propiedades de los objetos

han de ser modificadas. Puedes en consecuencia utilizar la tabla como guía de referencia para editar tu programa, y ella te permitirá comprobarlo si no funciona como es debido.

En la siguiente tabla se presentan los objetos del programa **Hola mundo** y sus propiedades.

Tipo de control	Propiedad	Valor
Form	Nombre Caption	frmHola Programa Hola Mundo
Command Button	Nombre Caption Font	cmdSalir &Salir Arial, Bold, 10
Command Button	Nombre Caption Font	cmdHola &Hola mundo Arial, Bold, 10
Command Button	Nombre Caption Font	cmdBorrar &Borrar Arial, Bold, 10
Text Box	Nombre Text Alignment Multiline Caption Font	CmdBorrar (Dejarlo en blanco)* 2 - Center True (Dejarlo en blanco) Arial, Bold, 10

Tabla 2.2

Nota: cualquier texto en la tabla cerrado entre paréntesis es una instrucción. Por ejemplo (Dejarlo en blanco) quiere decir que hay que borrar el contenido por defecto de la correspondiente propiedad.

3 Diagnósticos del sistema

En este capítulo tendremos el primer contacto con LEGO MindStorms. En este capítulo no montaremos ningún robot, pero por medio de un programa que será útil más adelante daremos inicio a la programación del RCX. Este programa, aun siendo muy sencillo, será muy útil cuando surja algún problema.

3.1. Contenidos de este capítulo

1. Controles del cuadro de herramientas: control Label.
2. Uso de constantes y variables en Visual Basic: introducción.
3. Estructuras de control: introducción. Estructura If ... Else ... Then de Visual Basic.
4. Programación con el control ActiveX Spirit.ocx.

3.2. Propuesta de proyecto

En la primera parte de esta práctica crearemos un programa que permita chequear el estado del RCX:

- Comprobará si la torre emisora de infrarrojos está conectada.
- Comprobará si la torre de infrarrojos se comunica con el RCX.
- Comprobará el nivel de carga de las baterías del RCX.

No hay que olvidar que la torre de infrarrojos (de aquí en adelante torre IR) utiliza una pila de 9V, y que su baja carga suele ser en ocasiones la razón de la falta de comunicación entre dicha torre y el RCX[2].

[2] La torre de infrarrojos correspondiente a las versiones 1.0 y 1.5 se conecta al puerto serie y utiliza una pila de 9V, mientras que la correspondiente a la versión 2.0 s conecta en un puerto USB, por lo cual no necesita otra fuente de alimentación.

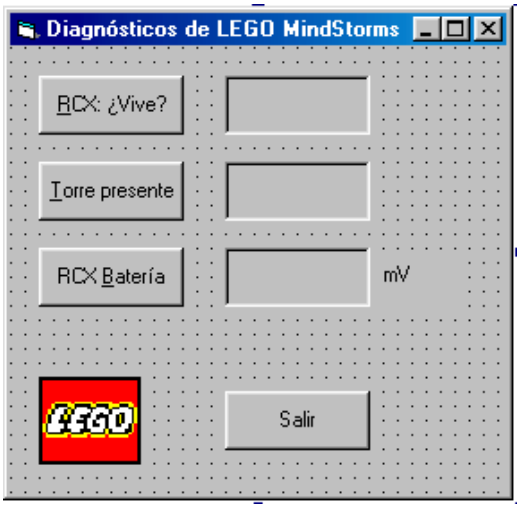


Figura 3.1:
este es el aspecto que tendrá el formulario una vez finalizado.

3.3. Uso de variables en Visual Basic

Para realizar las comprobaciones que requiere este programa, chequearemos el RCX. Los valores que el RCX devuelva habrá que almacenarlos para poder presentarlos posteriormente en el monitor. Para ello, utilizaremos las denominadas variables. Se llaman variables porque son objetos cuyos valores pueden ser modificados. Probablemente conocerás las variables utilizadas en matemáticas en expresiones como la siguiente:

$$x + y = 6$$

donde hay dos variables x e y.

Las variables en Visual Basic pueden también almacenar información no matemática. En el capítulo anterior hemos utilizado las siguientes expresiones:

```
txtHola.Text=""          y
txtHola.Text="Hola, ¿qué tal andamos?"
```

Lo que en realidad hemos hecho, ha sido dar a la propiedad txtHola.Text el valor "" y a continuación cambiarlo dándole el valor "Hola, ¿qué tal andamos?". La propiedad Text es un ejemplo de una variable, y el sufijo txtHola indica a Visual Basic que esta variable pertenece al objeto txtHola. De hecho, al ser todas las propiedades de un objeto modificables, también son variables. Podemos definir nuestras propias variables para usar en nuestros programas. Por ejemplo, si tuviésemos la expresión matemática

$$x + y = z$$

y damos a la variable x el valor 2, y a la variable y el valor 6, podríamos escribir un programa que calculase el valor de su suma, 8, y asignase ese valor a la variable z. Al dar un valor a una variable lo llamaremos asignación.

Hay diversos tipos de variables, aunque casi sólo vamos a utilizar en este manual las cadenas de texto (strings) y las variables numéricas. Sin embargo, tal y como habrás aprendido en matemáticas, hay distintos tipos de números: integer (números enteros como 1, 5, -33), números con punto flotante (1.233, -3.5, 8.0), números reales (6, π ,3½), etc. Utilizaremos el convenio de preceder el nombre de la variable por unas letras que indiquen la naturaleza de la variable que estamos utilizando. La siguiente tabla proporciona estos nombres convencionales y ejemplos de su uso.

Tipo de dato	Prefijo	Ejemplo
Bolean	bln	blnFound
Byte	byt	bytRasteData
Collection object	col	colWidgets
Currency	cur	curRevenue
Date (Time)	dtm	dtmInicio
Double	dbl	dblTolerancia
Error	err	errOrdenNum
Integer	int	intCantidad
Long	lng	lngDistancia
Object	obj	objActual
Single	sng	sngMedia
String	str	strNombre
User-defined type	udt	udtEmpleado
Variant	vnt	vntCheckSuma

Tabla 3.1

3.4. Constantes

Pero en ciertas ocasiones hay que utilizar valores que no varían, como pueden ser los números π y e. Para ello disponemos de las llamadas constantes. Las constantes son también ampliamente utilizadas en matemáticas y programación. La programación del Lego RCX puede simplificarse utilizando constantes predefinidas tales como MOTOR_A y TEMPO_2 (en el capítulo siguiente veremos como hacerlo).

3.5. El control Label

El control *Label* es un control gráfico que se puede utilizar para presentar texto que el usuario no pueda modificar directamente, aunque en la fase de diseño es posible escribir un código que pueda modificar el contenido del control *Label*.

Para crear un nuevo programa, hay que crear un proyecto nuevo.

- Iniciar Visual Basic. Si aparece la ventana *Nuevo Proyecto* hacer un doble clic sobre el icono *EXE estándar*. Si no aparece, seleccionar *Nuevo Proyecto* en el menú *Archivo*.
- Seleccionar *Componentes* en el menú *Proyecto* (otro modo de hacerlo es pulsando el botón derecho del ratón cuando el cursor se encuentra sobre el Cuadro de herramientas, y seleccionando a continuación *Componentes* en el menú contextual).

- Buscar “LEGO PbrickControl, OLE Control module” y marcar la casilla de verificación contigua. Pulsar Aceptar, con lo que el componente LEGO aparecerá en el Cuadro de herramientas.
- Hacer un doble clic sobre el icono de LEGO, con lo que el control LEGO aparecerá en el formulario. El nombre de este control es por defecto spirit1, pero puede ser modificado en la ventana de propiedades.
- Guardar el proyecto en el directorio C:\VBLego\Cap3. Para ello seleccionar *Guardar proyecto* en el menú *Archivo*. Una vez creado el directorio, guardar el proyecto y formulario con el nombre **Diagnósticos**.
- Diseña el formulario frmDiagnósticos a partir de los datos de la tabla 3.2.

Tipo de control	Propiedad	Valor
Form	Nombre Caption	frmDiagnósticos Diagnósticos de LEGO MindStorms
CommandButton	Nombre Caption ToolTipText	cmdRCXVive &RCX: ¿Vive? Chequea el estado del RCX
CommandButton	Nombre Caption ToolTipText	cmdTorreActiva &Torre presente Chequea el estado de la torre IR
CommandButton	Nombre Caption ToolTipText	cmdBatería RCX &Batería Voltaje de la batería
CommandButton	Nombre Caption	cmdSalir &Salir
Label	Nombre Alignement BorderStyle Caption	lblRCXVive 2 – Center 1 – Fixed Single (Dejar vacío)
Label	Nombre Alignement BorderStyle Caption	lblTorreActiva 2 – Center 1 – Fixed Single (Dejar vacío)
Label	Nombre Alignement BorderStyle Caption	lblBatería 2 – Center 1 – Fixed Single (Dejar vacío)

Tabla 3.2

Una vez finalizado el formulario deberá tener el aspecto que puede observarse en la figura 3.1.

- Introducir el código correspondiente a cmdSalir_Clic(), sin olvidar de insertar previamente la instrucción Option Explicit. Recordar que para introducir el código del procedimiento cmdSalir_Clic() hay que hacer un doble clic sobre el botón *Salir* en la vista *Objeto*.

```
 'Todas las variables han de ser declaradas

Option Explicit

Private Sub cmdSalir_Click()
    PBrickCtrl.CloseComm 'cierra el puerto de comunicaciones
    End
End Sub
```

- Introducir a continuación el código correspondiente al procedimiento Form_Load(). Para ello haz un doble-clic en cualquier lugar del formulario que no contenga controles.

```
Private Sub Form_Load()
    PBrickCtrl.InitComm 'abre el puerto de comunicaciones
End Sub
```

Analicemos este código en detalle.

Comentarios: la primera línea del código es un comentario. Un comentario es cualquier línea de texto precedida de un apostrofo (‘). Se puede escribir lo que se quiera después de un apóstrofo, ya que no será tenido en cuenta en la compilación del programa. Este texto, que aparecerá en color verde en el editor de Visual Basic, hará más comprensible el programa, sobre todo, cuando terceras personas tengan que leerlo.

Option Explicit: la instrucción Option Explicit establece que todas las variables que se quieran utilizar han de ser declaradas para poder ser utilizadas. Esto es muy útil, por que significa que si se comete un error al escribir el nombre de la variable, Visual Basic no asume que sea una nueva variable, sino que de hecho se ha cometido un error de escritura. Con el objeto de comunicarse con el RCX, el ordenador debe primero inicializar el puerto de comunicaciones serie. Esto se hace por medio de la orden PBrickCtrl.InitComm.

Abrir el puerto de comunicaciones: para que Visual Basic pueda comunicarse con el RCX hay que inicializar el puerto serie del ordenador Esto de hace por medio de la orden *PBrickCtrl.InitComm.* Para facilitar esta tarea conviene que este comando sea ejecutado inmediatamente tras el arranque del programa. Para ello hay que colocar la orden en procedimiento de evento *Private Sub Form_Load()*. Este procedimiento es inmediatamente después de cargar (abrir) el formulario. Para obtener el shell correspondiente a este procedimiento hacer un doble-clic sobre cualquier parte del formulario que no contenga controles.

Cerrar el puerto de comunicaciones: una vez completadas las comunicaciones con el RCX, la orden *PBrickCtrl.CloseComm* es llamada para cerrar el puerto serie. Dado que no interesa que esta orden se ejecute antes de dar por acabadas las comunicaciones con el RCX, lo mejor es colocarla en el procedimiento cmdSalir_Click(), que se ejecuta al finalizar el programa.

Sigamos ahora con el proyecto:

- Guarda el proyecto seleccionando *Guardar proyecto* en el menú *Archivo*.
- Ejecuta el programa por medio de un clic sobre el botón *Iniciar* (Play) de la barra de herramientas.
- Pulsar el botón *Salir*, y el programa finalizará.

El programa llama al procedimiento *InitComm* cuando el formulario es cargado y al procedimiento *CloseComm* cuando se pulsa el botón *Salir*. Y entre las llamadas a estas dos órdenes de sistema, se escribe el código que inicialice la interacción entre el RCX y la torre de infrarrojos.

3.6. Toma de decisiones

Las instrucciones para la toma de decisiones ofrecen al programa la posibilidad de elegir entre las opciones disponibles en el código y reaccionar apropiadamente a las situaciones que ocurran durante la ejecución. Con el objeto de poder tomar decisiones, se puede utilizar la estructura de control If ... Then ... Else.

3.6.1. Estructura de control If ... Then ... Else

Esta estructura de control tiene tres partes:

- **If** introduce la condición en la que se basará la decisión.
- **Then** identifica la acción que se ejecutará si la condición es verdadera.
- **Else** especifica la acción alternativa a ejecutar si la condición es falsa (es opcional).

Vamos a escribir el código necesario para obtener ciertos datos sobre el RCX:

- Introducir el resto del código del programa, empezando con el siguiente procedimiento:

```
Private Sub cmdBateria_Click()
    lblBateria.Caption=Str(PBrickCtrl.PBBattery)
End Sub
```

- Ahora añade este otro:

```
Private Sub cmdRCXVive_Click()
    If PBrickCtrl.PBAliveOrNot Then
            lblRCXVive.Caption="Sí"
      Else
            lblRCXVive.Caption="No"
      End If
End Sub
```

- Y para acabar el siguiente:

```
Private Sub cmdTorreActiva_Click()
    If PBrickCtrl.TowerAlive Then
            lblTorreActiva.Caption="Sí"
      Else
            lblTorreActiva.Caption="No"
      End If
End Sub
```

cmdBateria_Click(): por medio de este procedimiento podemos conocer la tensión de las pilas que alimentan el RCX. En el código, primero se lee el voltaje actual de la batería en milivoltios, dicho valor se convierte en una cadena de texto utilizando la función Str(), y se asigna dicho valor a la propiedad Caption del label lblBatería.

cmdRCXBizirik_Click(): en este procedimiento hemos utilizado la estructura de control de Visual Basic If ... Then ... Else. De este modo, podrá tomar la decisión adecuada en función de la respuesta que reciba del RCX. En este caso, la decisión será qué información ha de mostrar el formulario: si el RCX está encendido y la torre de infrarrojos puede conectarse con él, el label del resultado mostrará "Sí"; si no, mostrará "No". Adviértase que el final de la instrucción *If* ha de ser explícito, y para ello, se utiliza la instrucción *End If* a semejanza del final de las subrutinas con el End Sub.

El procedimiento cmdTorreActiva() comprueba si la torre de transmisión IR está activa. Si el hardware de la torre y la batería están funcionando, se visualizará un “Sí” en control label correspondiente. Si no, se visualizará un “No”.

cmdTorreActiva_Click(): este procedimiento comprueba si la torre de transmisión IR está activa. Si el hardware de la torre está como debe ser y su batería ofrece la tensión debida aparecerá la palabra “Sí” en el control. Si hay algún problema podremos leer “No”.

Métodos Spirit.ocx: en estos procedimientos utilizaremos tres métodos. El primero, *PBrickCtrl.PBBattery*, devuelve un valor numérico: la tensión de las pilas del RCX en milivoltios. Los otros dos devuelven valores lógicos, es decir, verdadero o falso; gracias a ello los podemos utilizar en estructuras de control para tomar decisiones.

Sigamos con el proyecto:

- Guarda el proyecto.
- Ejecuta el programa
- Con el RCX encendido cerca del transmisor de infrarrojos, pulsa los tres botones que conforman el test sucesivamente.
- Ahora desconecta el RCX y pulsa el botón *RCX ¿Vive?* (si el RCX está apagado, no pulses el botón “RCX Batería” ya que producirá un error).

El voltaje de las baterías es medido en milivoltios, y con nuevas baterías en el RCX, el voltaje debería ser cercano a los 9000mV. El valor decrece constantemente, por lo que se recomienda tener el RCX conectado sólo cuando es necesario. Para testear el alcance del transmisor de infrarrojos aléjalo poco a poco mientras pulsas el botón *¿RCX Activo*?.

Cuando se tienen dos o más RCX, se pueden producir interferencias en las transmisiones. Para evitarlo, se puede incluir en el programa una opción que especifique la potencia de transmisión al RCX. Con varios RCX en una misma habitación, debería estar configurado en corto alcance. De todos modos, conviene no tener encendido más de un RCX cuando se cargan los programas, por las razones que veremos más adelante.

- Añade los elementos de la tabla 3.3 al formulario, y el código que la sigue.

Tipo de control	Propiedad	Valor
CommandButton	Nombre Caption ToolTipText	cmdIRCorto IR &Corto Comunicaciones de corto alcance
CommandButton	Nombre Caption ToolTipText	cmdIRLargo IR &Largo Comunicaciones de largo alcance
Label	Nombre BorderStyle Caption	lblRango 1 – Fixed Single (Dejar vacío)

Tabla 3.3

```
Private Sub cmdIR_Click()
      PBrickCtrl.PBTxPower ALCANCE_CORTO
      LblAlcance= ” RCX configurado para alcance corto”
End Sub
```

```
Private Sub cmdIRLargo_Click()
      PBrickCtrl.PBTxPower ALCANCE_LARGO
      LblAlcance = ” RCX configurado para largo alcance”
End Sub
```

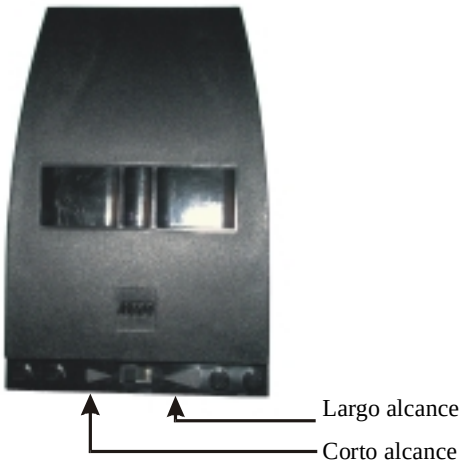


Figura 3.2
Configuración de la torre IR

Aquí estamos modificando la configuración de la potencia de transmisión del RCX. La potencia de transmisión de la torre de infrarrojos puede ser modificada manualmente por medio del interruptor de la parte delantera de la torre.

- Guarda el proyecto.
- Ejecuta el programa.
- Pulsa el botón de corto alcance de IR.
- Coloca el RCX a diferentes distancias de la torre (pero sin ocultarlo), y pulsar el botón *RCX ¿Vive?* a cada distancia. Por medio de la experimentación se puede establecer cual es el alcance cuando se configura para alcance corto.
- Pulsa el botón de largo alcance de IR
- Repite el procedimiento anterior para determinar el alcance cuando se configura para largo alcance.

Sea cual sea la potencia de transmisión del RCX que desee utilizar en otros programas con el RCX, hay que tomar la precaución de pulsar el botón correspondiente antes de salir del programa.

3.7. Mejora de la funcionalidad

Vamos a añadir algunas nuevas funcionalidades a este programa: una de ellas permitirá establecer la hora del RCX (la misma hora que en el ordenador), y la otra desconectar el RCX.

- Coloca los siguientes controles en el formulario:

Tipo de control	Propiedad	Valor
CommandButton	Nombre Caption ToolTipText	cmdSetHora Poner en &Hora Pone en hora el RCX (hora actual)
CommandButton	Nombre Caption ToolTipText	cmdRCXOff Apaga el R&CX Desconecta el RCX

Tabla 3.4

- Ahora introduce el código:

```
Private Sub cmdRCXOff_Click()
      PBrickCtrl.PBTurnOff
End Sub

Private Sub cmdSetHora_Click()
      PBrickCtrl.PBSetWatch Hour(Now),Minute(Now)
End Sub
```

El código para desconectar el RCX es bastante sencillo. El método denominado *PBTurnOff* es el que hace que el RCX se apague solo.

El segundo procedimiento no es tan sencillo. Se trata de dar al RCX la misma hora que la del ordenador. Primero hay que averiguar la hora del sistema, y para ello se utiliza la función *Now*. Cuando la función *Now* es llamada, recupera la fecha y hora del sistema, pero sólo necesitamos los valores de horas y minutos. Para obtener estos valores, se utilizan las funciones *Hour* y *Minute*. De este modo se pasan al método *SetWatch* los valores de la hora actual (valores entre 0 y 23) y el minuto actual (valores entre 0 y 59).

3.8. Ejercicio

La primera parte de esta práctica permite sondear el RCX para obtener información. Pero para ello, hay que pulsar los tres botones uno a uno. Ahora se pide escribir el código que permita obtener la misma información pulsando un solo botón.

Si el RCX no está presente, no hay que testear la batería, ya que se produciría un error. Escribe el código que evite esta situación.

4 Tu primer robot

4.1. Objetivos de este capítulo

1. Creación de plantillas en Visual Basic para evitar tareas repetitivas.
2. Robots móviles: control desde el ordenador por medio de formulario.
3. Uso de módulos: uso del módulo RCXdatos.bas para facilitar la lectura de los programas.
4. Cuadro de herramientas: barra de desplazamiento, botones de opción y frames.
5. Métodos que ofrece el RCX para controlar motores.

4.2. El robot

Hasta ahora el robot ha sido no móvil. Se puede dar movilidad a las construcciones utilizando los motores que vienen con el kit Lego. Para conectar los motores al RCX, se dispone de cables eléctricos especiales con conectadores del estilo ladrillo Lego.

Figura 4.1
Motor

Figura 4.2
Cable de conexión

El RCX tiene tres salidas para motores. Estos conectadores negros están etiquetados como A, B y C. Se puede conectar el otro extremo del cable al motor en cuatro posiciones diferentes. La orientación que elijas hará que el motor gire en sentido horario o antihorario.

Figura 4.3
Conexiones del RCX: en los conectadores grises de la parte superior se conectan los sensores y en los negros de la parte inferior los motores.

Para construir el robot que utilizaremos el robot que puede verse en la figura 4.4. Para montarlo hay que seguir las instrucciones que se dan en el apéndice A.

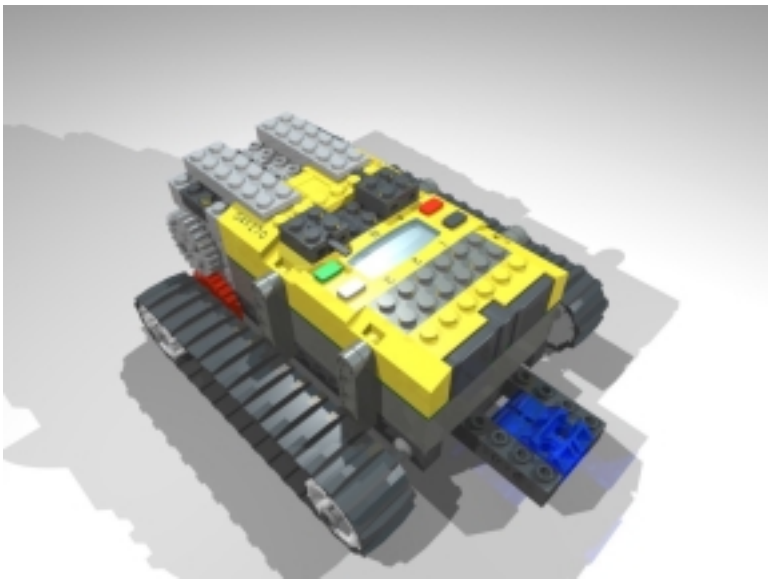


Figura 4.4
base del robot a utilizar

4.3. Plantilla

En cada capítulo estamos creando un nuevo proyecto y, tal y como veremos, ciertos pasos se repiten una y otra vez, por ejemplo, colocar el componente spirit.ocx en el cuadro de herramientas y en el formulario. Del mismo modo que sucede con otras aplicaciones informáticas, aquí también podemos utilizar plantillas para evitar repetir las labores iniciales de configuración.

Utilizar esta plantilla ofrece las siguientes ventajas:

- El componente Spirit.ocx estará disponible desde el principio.
- Los argumentos que utilizarán los controles del spirit.ocx serán más fáciles de entender.

La plantilla va a ser lo más básica posible, pero es posible incorporar otros elementos que también se repitan en diferentes programas (por ejemplo, los procedimientos de entrada y salida).

Dado que será necesario en la plantilla veamos que es el módulo *RCXdatos.*

4.3.1. Módulo RCXdatos

Los argumentos de los métodos del control ActiveX Spirit.ocx son números nada significativos. Para facilitar la edición de programas utilizaremos constantes que sustituirán los números por nombres más expresivos. Algunas serán comunes para todos los programas y se guardan en el módulo RCXdatos.bas. Sin embargo, otras las nombraremos en cada programa. En el apéndice B se recoge el contenido de este módulo que se ofrece junto a este manual, ya que será el que utilizaremos de aquí en adelante.

Si lo deseas puedes adaptarlo a tu gusto.

4.3.2. Procedimiento

Veamos ahora el procedimiento a seguir para crear la plantilla:

- Iniciar Visual Basic. Si aparece la ventana *Nuevo Proyecto* hacer un doble clic sobre el icono *EXE estándar*. Si no aparece, seleccionar *Nuevo Proyecto* en el menú *Archivo*.
- Para instalar Spirit.ocx en Visual Basic, seleccionar *Componentes* en el menú *Proyecto* (otro modo de hacerlo es pulsando el botón derecho del ratón cuando el cursor se encuentra sobre el Cuadro de herramientas, y seleccionando a continuación *Componentes* en el menú contextual).

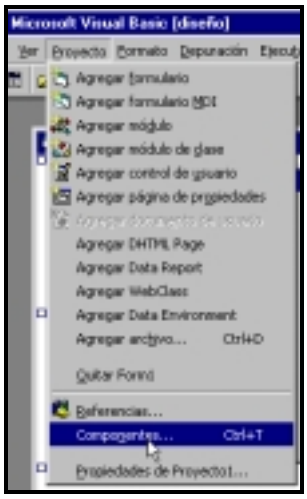


Figura 4.5
Selecciona *Componentes* (Ctrl+T) en el menú *Proyecto*.

- Busca "*LEGO PbrickControl, OLE Control module*" y marca la casilla de verificación contigua. Pulsar Aceptar, con lo que el logo LEGO aparecerá en el Cuadro de herramientas. Si no apareciese, utilizar *Examinar* para buscarlo.

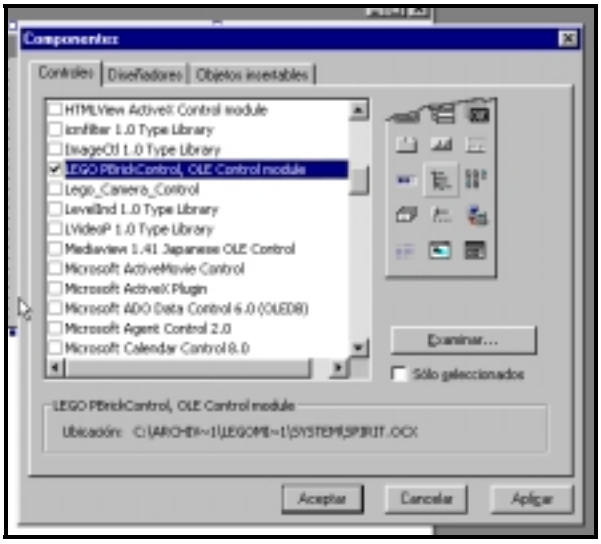


Figura 4.6
Busca el control ActiveX de Lego en la lista de componentes.

- Haz un doble clic sobre el icono de LEGO, con lo que el control LEGO aparecerá en el formulario. Dale las dimensiones deseadas y colócalo en una esquina del formulario.

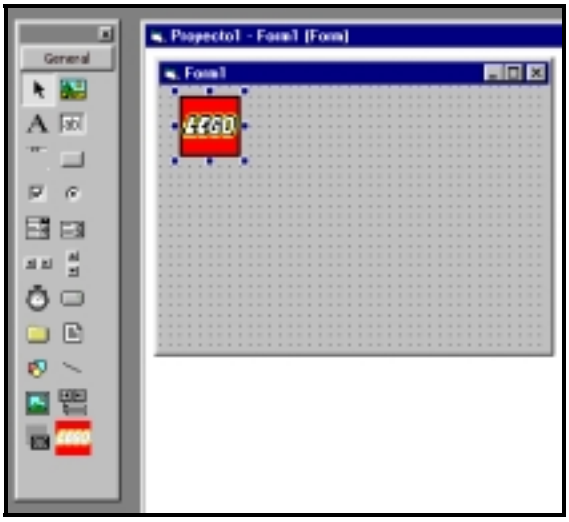


Figura 4.7
Coloca el control LEGO en el formulario.

Seleccionando el objeto, ciertas propiedades del control Spirit.ocx pueden ser modificadas. LEGO recomienda utilizar como nombre del objeto **PbrickCtrl**, pero puedes utilizar cualquier otro. Si deseas modificarlo sigue el siguiente paso:

- Tras seleccionar el objeto LEGO, haz clic sobre la propiedad (Nombre) y escribe **PbrickCtrl**.

El resto de propiedades están configuradas por defecto para trabajar con el RCX 1.0 y 1.5 (utiliza el puerto de comunicaciones serie COM1). Para programar para CyberMaster hay que hacer alguna modificación.

Ahora le añadiremos el módulo RCXdatos.bas:

- Elige *Agregar módulo* en el menú *Proyecto*.
- Haz clic sobre *Existente*, y localiza RCXdatos.bas, archivo que has obtenido junto a este manual.

- Cuando lo encuentres, selecciónalo y pulsa el botón *Abrir*.

La ventana *Proyecto* aparecerá tal y como se ve en la figura 4.6 (si tienes carpetas en la ventana *Proyecto*, haz clic sobre ellas para ver lo que contienen).

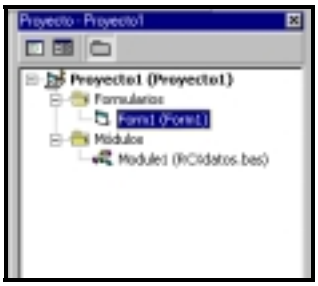


Figura 4.8
Ventana *Proyecto*

- Selecciona Form1 en la ventana *Proyecto*.
- Selecciona *Guardar Form1* Como en el menú *Archivo*.
- Localiza el directorio C:\Archivos de programa\Microsoft Visual Studio\Vb98\Template\Projects[3]

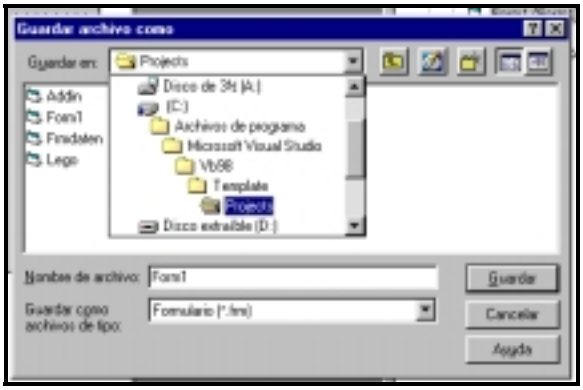


Figura 4.9
Busca el directorio *Projects* de Visual Basic

- Nombra el formulario como **Lego** y haz clic en el botón *Guardar*.

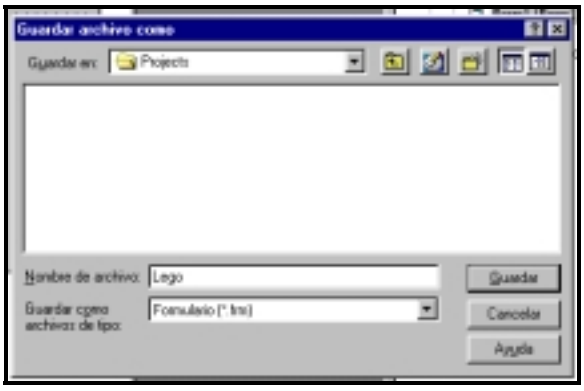


Figura 4.10
Guarda el formulario con el nombre **Lego**

- Selecciona *Guardar proyecto como* en el menú *Archivo* y en el cuadro de texto del nombre escribe **Lego**.
- Haz clic en el botón *Guardar*
- Haz clic para seleccionar el Módulo1 (RCXdatos.bas) en la ventana *Proyecto*.

[3] Dependiendo de la versión puede haber alguna diferencia en la ruta del directorio *Projects*

- Dale como nombre **RCXdatos**.
- Pulsa el botón *Guardar*.
- Selecciona Salir en el menú *Archivo* para salir de Visual Basic.
- Inicia Visual Basic otra vez.

Ahora al reiniciar Visual Basic aparecerá el cuadro de diálogo 4.10. En él encontrarás el nuevo icono Lego.

Figura 4.11
Ahora el cuadro de diálogo *Nuevo proyecto* ofrece la plantilla **Lego**

- Si no aparece el cuadro de diálogo, selecciona *Nuevo proyecto* en el menú *Archivo*.

Ahora puedes utilizar el icono Lego para iniciar tus proyectos. Puedes incorporar otras prestaciones a la plantilla si así lo deseas, por ejemplo, el código necesario para el control de errores en la transferencia de programas del ordenador al RCX.

4.4. Propuesta de trabajo

En este capítulo vamos a desarrollar un programa que nos permita controlar un robot móvil desde el ordenador. Por medio de este programa podremos guiar el robot por diversos lugares (adelante, atrás giro a la izquierda o a la derecha).

En la segunda parte de este capítulo ampliaremos las capacidades del programa de tal modo que podremos controlar la potencia de los motores.

4.5. Programa

En el capítulo anterior hemos aprendido cómo utilizar el control Spirit.ocx para comunicarse con el RCX. Vamos a crear ahora un programa que controle un automóvil que haremos utilizando el kit RIS.

Para crear el programa sigamos los siguientes pasos:

- Iniciar Visual Basic. Si aparece la ventana *Nuevo Proyecto* hacer un doble clic sobre el icono *Lego*[4]. Si no aparece, seleccionar *Nuevo Proyecto* en el menú *Archivo*.
- Asegurarse de que la ventana Form1 del nuevo proyecto está seleccionada, y seleccionar *Guardar Form1 Como* del menú *Archivo*.
- Por medio del cuadro de diálogo *Guardar Como*, localizar el directorio C:\VBLEGO\.
- Haz un clic sobre el botón *Crear Nueva Carpeta*, y darle el nombre Cap4.
- Abrir el recién creado directorio.
- Nombrar el formulario como *Control Remoto* y pulsar el botón *Guardar*.
- Selecciona *Guardar proyecto como* en el menú *Archivo*.
- El primer archivo que será guardado será el archivo .bas. Introducir el nombre **Remoto** y pulsar el botón *Guardar*.
- A continuación pedirá el nombre del archivo .vbp. Llamarlo también **Remoto** y pulsar a continuación el botón *Guardar*.
- Diseña el formulario frmRemoto de acuerdo con los datos de la tabla 4.1. Cuando lo acabes, el resultado habrá de tener el aspecto que se puede ver en la figura 4.12.

Figura 4.12
el primer paso de este proyecto es diseñar un formulario como este

[4] A partir de ahora utilizaremos esta plantilla que ofrece como punto de partida un formulario con el componente spirit.ocx incluido. En el apéndice B se describe como crear esta plantilla.

Tipo de control	Propiedad	Valor
Form	Nombre Caption	frmRemoto Control Remoto
CommandButton	Nombre Caption ToolTipText	cmdAdelante &Adelante Movimiento de avance
CommandButton	Nombre Caption ToolTipText	cmdAtrás A&trás Movimiento de retroceso
CommandButton	Nombre Caption ToolTipText	cmdIzq &Izquierda Giro a la izquierda
CommandButton	Nombre Caption ToolTipText	cmdDcha &Derecha Giro a la derecha
CommandButton	Nombre Caption ToolTipText	cmdSalir &Salir Finalizar el programa

Tabla 4.1

Hasta ahora el único tipo de evento que hemos utilizado con los botones ha sido el evento *Clic* (cuando se hace clic sobre algún botón sucede algo). En este proyecto si queremos que el robot se mueva hacia un lado u otro no haremos clic sobre un botón. Al pulsar el botón Adelante el vehículo se moverá hacia delante y al soltarlo se detendrá. Lo mismo sucederá con el resto de los botones.

La edición de eventos diferentes a Clic es algo diferente. Hasta ahora para introducir el código correspondiente de un botón de comando hacíamos un doble clic sobre el botón, y un shell del procedimiento era creado. Pero el shell creado de esta manera sólo abarca el evento Clic, y no los eventos MouseUp o MouseDown que vamos a implementar ahora.

Vamos a crear el código correspondiente a la orden cmdAdelante_MouseDown:

- Hacer un doble clic sobre el botón cmdAdelante del formulario del modo habitual.

Tenemos ahora delante la vista *Código*, como en el capítulo anterior. En las dos listas desplegables de la parte superior de la ventana debe estar presente cdmAdelante en la de la izquierda (lista de *Objetos*) y Clic en la de la derecha (lista de *Procedimientos*).

- Haz un click en la flecha hacia debajo del cuadro de la derecha y selecciona la opción *MouseDown*.

Un nuevo shell es creado para este evento.

- Para eliminar el evento cmdAdelante_Click() selecciónalo y bórralo.
- Ahora hay que introducir el código en el shell del procedimiento que acabamos de crear

```
Private Sub cmdAdelante_MouseDown(Button As Integer, Shift As Integer, _
X As Single, Y As Single)
      PBrickCtrl.SetFwd MOTOR_A + MOTOR_C
```

```
        PBrickCtrl.On MOTOR_A + MOTOR_C 'Avanza
End Sub
```

En la primera línea del código anterior, el guión bajo es usado para finalizar la línea. Sin embargo, ese no es el final de la línea de código. El carácter guión bajo notifica a Visual Basic que la línea de código no ha finalizado todavía y que sigue en la siguiente línea. Esto es útil ya que a veces hay que escribir largas líneas de código, como en el procedimiento anterior.

- A continuación selecciona MouseUp en el cuadro combo *Procedimientos*, y escribe el siguiente código::

```
Private Sub CmdAdelante_MouseUp(Button As Integer, Shift As Integer, X _
As Single, Y As Single)
    PBrickCtrl.Off MOTOR_A + MOTOR_C
End Sub
```

- Utilizando el mismo método, editar el siguiente código:

```
Option Explicit

Private Sub Form_Load()
    PBrickCtrl.InitComm             'Inicializa el puerto serie del PC
    PBrickCtrl.SetPower MOTOR_A + MOTOR_C, CTE, 2
End Sub

Private Sub cmdIzq_MouseDown(Button As Integer, Shift As Integer, X _
As Single, Y As Single)
    PBrickCtrl.SetFwd MOTOR_C
    PBrickCtrl.On MOTOR_C
End Sub

Private Sub cmdIzq_MouseUp(Button As Integer, Shift As Integer, X _
As Single, Y As Single)
    PBrickCtrl.Off MOTOR_C
End Sub

Private Sub cmdSalir_Click()
    PBrickCtrl.CloseComm
    End
End Sub
```

4.5.1. Descripción del programa

De la misma manera que en el capítulo anterior, el método InitComm es llamado en el inicio por el procedimiento *Form_Load*. La instrucción

Tal y como pudimos ver en el capítulo anterior, al iniciarse el programa se ejecuta el procedimiento *Form_Load*. Este procedimiento cumplirá dos funciones: abrir el puerto de comunicaciones (InitComm) y establecerla potencia de los motores.

Potencia de los motores: para establecer la potencia de los motores utilizaremos la siguiente línea de código:

PBrickCtrl.SetPower MOTOR_A + MOTOR_C, CTE, 2

La potencia es fijada al valor constante (CTE) 2. La potencia puede tomar un valor entre 0 y 7: 7 es el valor correspondiente a la mayor potencia y 0 a la menor (el nivel de potencia 0 no significa que el motor de detenga). En este caso el nivel de potencia de los dos motores será 2.

Esto no tiene un efecto apreciable en la velocidad del motor, pero sí en la potencia. Cuando el robot circula sobre una superficie con alto rozamiento como puede ser una alfombra, hay que darle un alto valor.

Cuando el botón *cmdAdelante* es presionado, el robot avanza. El procedimiento de evento:

```
Private Sub cmdAdelante_MouseDown(Button As Integer, Shift As Integer, _
X As Single, Y As Single)
      PBrickCtrl.SetFwd MOTOR_A + MOTOR_C
      PBrickCtrl.On MOTOR_A + MOTOR_C          'Avanza
End Sub
```

es ejecutado cuando el botón está presionado.

Los dos motores se configuran con movimiento de avance (SetFwd), y a continuación se conectan(On).

Cuando se suelta el botón, el procedimiento de evento

```
Private Sub cmdAdelante_MouseUp(Button As Integer, Shift As Integer, X _
As Single, Y As Single)
    PBrickCtrl.Off MOTOR_C
End Sub
```

es ejecutado, con lo que los motores se detienen.

El código para girar a la izquierda es similar, pero en este caso lo haremos haciendo avanzar únicamente al motor derecho. El método SetFwd hace que el motor se mueva en dirección de avance. Otros posibles métodos para modificar la dirección son los siguientes:

.SetRwd - hace que el motor especificado se mueva hacia atrás.

.AlterDir - cambia la dirección del motor especificado.

4.5.2. Ejercicio

Falta todavía el código para que el robot se mueva hacia atrás o gire hacia la derecha. Edita dicho código copiando el código correspondiente al movimiento de avance y giro a la izquierda para modificarlo a continuación.

4.6. Uso de gráficos en los botones de comando

Hasta ahora, hemos visto cómo colocar un texto sobre los botones de comando por medio de la propiedad *Caption*. Si se desea, es posible sustituir dichos textos por gráficos, lo cual es muy útil en ciertos programas. Para ello hay que seguir los siguientes pasos:

- Selecciona el botón de comando que deseas modificar.
- Borra el contenido de la propiedad *Caption* (si es que lo tiene).
- Cambia la propiedad *Style* a **1 – Graphical**.
- Por medio de la propiedad *Picture*, localiza el archivo gráfico que deseas utilizar.

En este ejercicio los autores han utilizado imágenes contenidas en el directorio Microsoft Visual Studio/Common/Graphics/Icons/Arrows, aunque este directorio puede no existir en tu ordenador, ya que depende de cómo se ha hecho la instalación inicial.

4.7. Mejora del control del robot

Vamos a ampliar la potencialidad de nuestro programa. Cuando hemos creado el evento Form_Load() hemos establecido un valor de potencia para los motores, que ha permanecido invariable durante la ejecución del programa. Vamos a hacer algunos cambios en el programa para que podamos modificar dicha potencia cuando queramos. Lo más adecuado para ello, puede ser utilizando una barra de desplazamiento horizontal. Es el icono *HScrollBar* (el nombre aparece

cuando detenemos el cursor sobre él). Un cuadro de texto mostrará el valor de la velocidad (no será la velocidad verdadera).

En el formulario Control Remoto del programa anterior, vamos a colocar una barra de desplazamiento:

- Seleciona el control *Barra de Desplazamiento Horizontal* ( )de la caja de herramientas y coloca el cursor sobre el formulario (el cursor tomará la forma de una cruz). Presiona el botón izquierdo del ratón y arrastra el ratón hasta que se forme un rectángulo del tamaño deseado. Suelta el botón del ratón y el control será visualizado. Puedes modificar el tamaño de la barra de desplazamiento seleccionándola y estirando de cualquiera de los puntos azules del contorno.

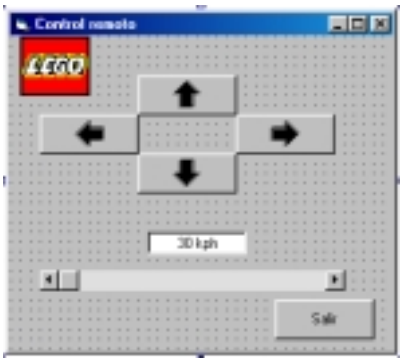


Figura 4.13
formulario con barra de desplazamiento y cuadro de texto.

Tipo de control	Propiedad	Valor
Barra horizontal de desplazamiento	Nombre Max Min LargeChange SmallChange Value	hsbVelocidad 7 0 1 1 2
Cuadro de texto Text Box	Nombre Value Alignment	txtVelocidad 20 km/h Center

Tabla 4.2

- Haz un doble clic sobre la barra de desplazamiento.

```
Private Sub hsbVelocidad_Change()
    PBrickCtrl.SetPower MOTOR_A + MOTOR_C, CTE, hsbVelocidad.Value
    TxtVelocidad.Text = Str(hsbVelocidad.Value * 10 + 10) + "km/h"
End Sub
```

4.7.1. Descripción del código

La barra de desplazamiento horizontal comprende los valores entre 0 y 7 (Min – Max). La configuración actual está contenida en la propiedad *hsbVelocidad.Value*. La instrucción

```
PBrickCtrl.SetPower MOTOR_A + MOTOR_C, CTE, hsbVelocidad.Value
```

establece como potencia de los motores el valor actual (hsbVelocidad.Value) de la barra de desplazamiento.

La siguiente línea del procedimiento

```
TxtVelocidad.Text = Str(hsbVelocidad.Value * 10 + 10) + "km/h"
```

multiplica la propiedad *hsbVelocidad.Value* por 10 con lo que obtenemos un valor de velocidad (no real). A este valor le sumamos 10. La razón es que el valor cero de potencia no quiere decir que esté parado, sino que es el valor más bajo de todos, por lo que con un valor cero puede haber movimiento. La función Str convierte el valor resultante de las anteriores operaciones en una cadena de texto, ya que el contenido de la propiedad Text de un cuadro de texto ha de ser una cadena de texto. Para acabar añade a la anterior cadena de texto las letras km/h. Adviértase que los datos tipo texto se escriben entre comillas.

4.8. Más mejoras

Nuestro programa funciona bien, pero nos obliga a que los motores siempre estén conectados siempre en las salidas A y C (es decir, 0 y 2). Vamos a transformarlo en un programa un poco más flexible, dando la oportunidad al usuario de especificar donde ha conectado los motores.

Para ello utilizaremos dos nuevos controles: botones de opción y frames.

4.8.1. Botones de opción (OptionButton)

Un control *OptionButton* () visualiza una opción que sólo puede ser sí o no. Si colocas botones de opción en un formulario y ejecutas el programa, varios botones de opción suelen estar asociados entre sí, y sólo se puede seleccionar uno a la vez. Sin embargo, a veces son necesarios dos o más grupos de botones de opción en el mismo formulario. Esto lo puedes hacer utilizando *Frames*, que permiten al programa distinguir entre distintos grupos de botones de opción.

4.8.2. Control Frame

Un control *Frame* proporciona una agrupación identificable para los controles. Puedes utilizar un *Frame* para subdividir funcionalmente un formulario, por ejemplo, para separar grupos de botones de opción, como vamos a hacer aquí.

- Para agrupar controles, dibuja primero un control *Frame* (el icono con “xy” en la esquina superior izquierda ), y a continuación dibuja los controles en el interior del *Frame*.
- Recuerda siempre dibujar primero el control *Frame* antes de colocar cualquier botón de opción. Dibuja los botones de opción del motor izquierdo en el *Frame* de la izquierda, y los del derecho en el de la derecha

Nota: para seleccionar múltiples controles en un formulario, mantén pulsada la tecla <**Ctrl**> mientras los seleccionas con el ratón. Puedes ir entonces a la ventana propiedades, y dar a todos ellos las mismas propiedades, por ejemplo, fuente o color.

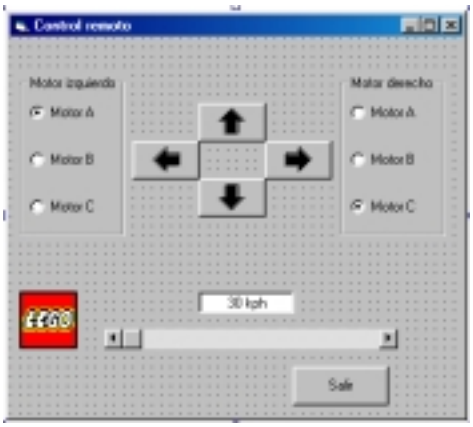


Figura 4.14
el formulario acabado

4.8.3. Código

Tipo de control	Propiedad	Valor
Frame	Nombre Caption	fraIzquierdo Motor Izquierdo
Option Button	Nombre Caption Value	optIzquierdoA Motor A True
Option Button	Nombre Caption	optIzquierdoB Motor B
Option Button	Nombre Caption	optIzquierdoC Motor C
Frame	Nombre Caption	fraDerecho Motor Derecho
Option Button	Nombre Caption	optDerechoA Motor A
Option Button	Nombre Caption	optDerechoB Motor B
Option Button	Nombre Caption Value	optDerechoC Motor C True

Tabla 4.3

A continuación se recoge el código al completo:

```
Option Explicit
Dim strMotorIzquierdo, strMotorDerecho As String

Private Sub Form_Load()
    PBrickCtrl.InitComm                    'inicia el puerto serie del PC
    strMotorIzquierdo = MOTOR_A            'motor A como opción inicial
    strMotorDerecho = MOTOR_C              'motor C como opción inicial
    PBrickCtrl.SetPower strMotorIzquierdo + strMotorDerecho, 2, 2
End Sub

Private Sub cmdAdelante_MouseDown(Button As Integer, Shift As Integer, _
X As Single, Y As Single)
    PBrickCtrl.SetFwd strMotorIzquierdo + strMotorDerecho
```

```
    PBrickCtrl.On strMotorIzquierdo + strMotorDerecho
End Sub

Private Sub cmdAdelante_MouseUp(Button As Integer, Shift As Integer, X _
As Single, Y As Single)
    PBrickCtrl.Off strMotorIzquierdo + strMotorDerecho
End Sub

Private Sub cmdAtrás_MouseDown(Button As Integer, Shift As Integer, X _
As Single, Y As Single)
    PBrickCtrl.SetRwd strMotorIzquierdo + strMotorDerecho
    PBrickCtrl.On strMotorIzquierdo + strMotorDerecho
End Sub

Private Sub cmdAtrás_MouseUp(Button As Integer, Shift As Integer, X _
As Single, Y As Single)
    PBrickCtrl.Off strMotorIzquierdo + strMotorDerecho
End Sub

Private Sub cmdDcha_MouseDown(Button As Integer, Shift As Integer, X _
As Single, Y As Single)
    PBrickCtrl.SetFwd strMotorIzquierdo
    PBrickCtrl.On strMotorIzquierdo
End Sub

Private Sub cmdDcha_MouseUp(Button As Integer, Shift As Integer, X _
As Single, Y As Single)
    PBrickCtrl.Off strMotorIzquierdo
End Sub

Private Sub cmdIzq_MouseDown(Button As Integer, Shift As Integer, X _
As Single, Y As Single)
    PBrickCtrl.SetFwd strMotorDerecho
    PBrickCtrl.On strMotorDerecho
End Sub

Private Sub cmdIzq_MouseUp(Button As Integer, Shift As Integer, X _
As Single, Y As Single)
    PBrickCtrl.Off strMotorDerecho
End Sub

Private Sub hsbVelocidad_Change()
    PBrickCtrl.SetPower strMotorIzquierdo + strMotorDerecho, 2,_
hsbVelocidad.Value
    txtVelocidad = Str(hsbVelocidad.Value * 10 + 10) + "km/h"
End Sub

Private Sub optDerechoA_Click()
    strMotorDerecho = MOTOR_A                                   'Motor A
End Sub

Private Sub optDerechoB_Click()
    strMotorDerecho = MOTOR_B                                   'Motor B
End Sub

Private Sub optDerechoC_Click()
    strMotorDerecho = MOTOR_C                                   'Motor C
End Sub

Private Sub optIzquierdoA_Click()
```

```
    strMotorIzquierdo = MOTOR_A
      'Motor A
End Sub

Private Sub optIzquierdoB_Click()
    strMotorIzquierdo = MOTOR_B                                    'Motor B
End Sub
Private Sub optIzquierdoC_Click()
    strMotorIzquierdo = MOTOR_C
      'Motor C
End Sub


Private Sub cmdSalir_Click()
    PBrickCtrl.CloseComm
    End
End Sub
```

4.8.4. Descripción del programa

La instrucción:

```
Option Explicit
Dim strMotorIzquierdo, strMotorDerecho As String
```

declara dos variables que guardarán cadenas de texto (strings).

En el procedimiento de evento Form_Load se asigna el valor MOTOR_A a la variable *strMotorIzquierdo*, y el valor MOTOR_C a la variable *strMotorDerecho*. La razón de que sea así la puedes deducir observando detenidamente la tabla 3.3. El valor del botón de opción *optIzquierdaA* es *True* (verdadero), lo que significa que este es el botón seleccionado cuando el programa se inicia. De este modo el motor inicialmente seleccionado y el valor visualizado por medio del botón de opción serán el mismo. Lo mismo sucede con el motor derecho (el valor por defecto es *optDerechoC*).

En el código anterior hemos utilizado las constantes MOTOR_A y MOTOR_C. En este han sido reemplazadas por las variables *strMotorIzquierdo* y *strMotorDerecho* respectivamente.

El procedimiento de evento

```
Private Sub optDerechoA_Click()
    strMotorDerecho = "0"                                    'Motor A
End Sub
```

se ejecuta cuando se hace clic en el botón de opción *optDerechoA*. La consecuencia es que se asigna el valor 0 (la constante a la que sustituye MOTOR_A) a la variable *strMotorDerecho* (el motor conectado a la salida A quedará configurado como motor derecho).

4.8.5. Ejecución del programa

- Guarda y ejecuta el programa:
- Conecta los motores en diferentes salidas, y selecciona las salidas por medio de los botones de opción para reconfigurarlos.
- Guía el robot por medio de los controles del programa.

Habrá advertido que el movimiento de la barra de desplazamiento genera consecuencias inesperadas cuando se arrastra la barra en lugar de utilizar las flechas laterales (el valor en el cuadro de texto no cambia hasta que sueltas el botón del ratón). Para remediarlo, introduce el código siguiente:

- En el cuadro desplegable *Objeto* de la parte superior de la ventana *Código*, selecciona *hsbVelocidad*.
- En el cuadro desplegable *Procedures*, selecciona *Scroll*.

Un shell para el procedimiento aparecerá:

```
Private Sub hsbVelocidad_Scroll()
    PBrickCtrl.SetPower strMotorIzquierdo + strMotorDerecho, 2,_
      hsbVelocidad.Value
    txtVelocidad = Str(hsbVelocidad.Value * 10 + 10) + "Km/h"
End Sub
```

5 Uso de los sensores

5.1. Contenidos de este capítulo

Este capítulo trata del aparato sensor de los robots, pero todavía sin relacionarlo con el aparato motor. Una vez configurados los sensores obtendremos sus lecturas y las presentaremos en un formulario.

1. Uso de sensores: tipos y modos de sensores.
2. Sensor de contacto y sensor de luz.
3. Presentación de las lecturas de los sensores en formularios.
4. Cuadro de herramientas: temporizador (Timer), figura geométricas (Shape) y listas (Combo-box y List-Box).

5.2. Aparato sensorial

Además de tener la capacidad de controlar salidas, como los motores, el RCX puede recibir entradas externas de sensores. Hay varios tipos de sensores que pueden ser utilizados con el RCX: sensores activos (los que necesitan alimentación para poder funcionar como el sensor de luz) y los sensores pasivos (sensor de contacto). El kit básico Lego MindStorms suministra dos sensores de contacto y uno de luz. El kit Ultimate Accesory kit contiene un sensor angular y por medio del servicio al consumidor de LEGO pueden adquirirse sensores de temperatura. A partir de ahí cada uno puede diseñar sus propios sensores.

.Al contrario que en el caso de los motores, la orientación de los conectadores en el sensor no marca diferencias, y el sensor de luz tiene su propio cable.

Para poder programar el uso de los sensores en Visual Basic, primero han de ser configurados. El tipo de sensor utilizado y el formato en que se desea obtener las lecturas han de ser establecidos antes de sondear (leer) el sensor.

Figura 5.1: sensor de luz Sensor de contacto Cable conectador

5.2.1. Sensor de contacto

Vamos a configurar el sensor de contacto.

Para crear un nuevo programa, has de crear un nuevo proyecto:

- Iniciar Visual Basic. Si aparece la ventana *Nuevo Proyecto* hacer un doble clic sobre el icono *Lego*. Si no aparece, hazlo seleccionando *Nuevo Proyecto* en el menú *Archivo*.
- Tal y como has hecho anteriormente, guarda todos los nuevos archivos con el nombre **Sensores**. Selecciona C:\VBLEGO\cap05 como destino para guardar el proyecto.
- Diseña el formulario *frmSensores* de acuerdo con la tabla 5.1.

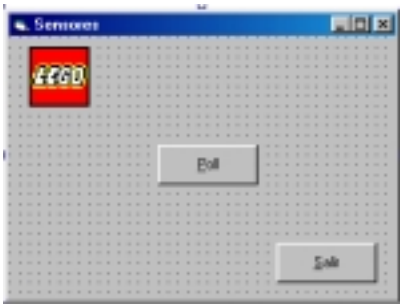


Figura 5.2
Comienza creando este sencillo formulario

Tipo de control	Propiedad	Valor
Form	Nombre Caption	frmSensores Sensores
CommandButton	Nombre Caption	cmdPoll &Poll
CommandButton	Nombre Caption	cmdSalir &Salir
Cuadro de texto	Nombre Alignment Caption	txtPoll 2 – Center (Dejarlo en blanco)

Tabla 5.1

- Introduce el siguiente código:

```
'Todas las variables han de ser declaradas
Option Explicit

Private Sub cmdSalir_Click()
    PBrickCtrl.CloseComm
    End
End Sub

Private Sub cmdPoll_Click()
    'configura la salida 1 como sensor de contacto
    PBrickCtrl.SetSensorType SENSOR_1, TIPO_SWITCH
    'da al cuadro de texto la lectura del sensor 1
    txtPoll.Text = PBrickCtrl.Poll(SENVAL, SENSOR_1)
End Sub
```

```
Private Sub Form_Load()
    PBrickCtrl.InitComm
End Sub
```

5.2.2. Descripción del programa

El procedimiento de evento cmdPoll_Click() coloca en el cuadro de texto txtPoll el valor actual (como un valor booleano, es decir, verdadero o falso) del sensor conectado en la entrada 1.

```
PBrickCtrl.SetSensorType SENSOR_1, TIPO_SWITCH
```

Esta línea indica que debería haber conectado un sensor de contacto en la entrada 1, y que has configurado dicho sensor como *Switch* (interruptor). Puedes también configurar las entradas SENSOR_2 y SENSOR_3. Los posibles tipos de sensores con sus valores numéricos y constante correspondientes son los siguientes:

Tipo de sensor	Constante	Número
Ninguno	NO_TIPO	0
Contacto	TIPO_SWITCH	1
Temperatura	TIPO_TEMP	2
Luz	TIPO_LUZ	3
Angular	TIPO_ANGULO	4

Tabla 5.2

El sensor ya está correctamente configurado, con lo que es posible sondearlo.

```
txtPoll.Text = PBrickCtrl.Poll(SENVAL, SENSOR_1)
```

Por medio de esta instrucción vamos a hacer que la lectura actual (SENVAL) del sensor 1 se asigne al cuadro de texto txtPoll (valor booleano, es decir, 1 si es verdadero y 0 si es falso).

El método Poll puede ser usado para obtener diferentes tipos de informaciones del RCX. El primer parámetro indica que dato quieres obtener, en tu caso el valor del sensor (SENVAL) y el segundo parámetro indica qué sensor quieres sondear, en este caso el sensor 1.

Nota: el valor del segundo parámetro depende del tipo de dato, por ejemplo, si el origen es VAR, una variable, el segundo parámetro será un número entre 0 y 31. Los posibles valores se recogen en la tabla 5.3..

Origen del dato	Constante	Número	Descripción
0	VAR	0-31	Variable 0 – 31
1	TEMP	0-3	Temporizador 0 – 3
2	CTE	-	-
3	MOTEST	0, 1, 2	Estado del motor. La información está empaquetada: Bit 7: ON/OFF 1/0 Bit 6: Freno/Float 1/0 Bit 5: Salida Nº HiBit Bit 4: Salida Nº LoBit Bit 3: Dirección CW/CCW5 1/0 Bit 2: Nivel de potencia: bit más significativo Bit 1: Nivel de potencia: Bit 0: Nivel de potencia: bit menos significativo
4	RAN	-	-
8	KEYS	-	Nº de programa, es decir, programa seleccionado actualmente
9	SENVAL	0, 1, 2	Valor del sensor. Valor medido en la entrada. Depende del modo actual de operación.
10	SENTIPO	0, 1, 2	Tipo de sensor. Comunica para qué tipo de sensor está configurada la entrada.
11	SENMODO	0, 1, 2	Modo de sensor. Comunica para qué modo de sensor está configurada la entrada.
12	SENRAW	0, 1, 2	Sensor Raw, es decir, el valor análogo al medido en la entrada.
13	BOOL	0, 1, 2	Sensor Booleano. Devuelve el valor booleano de la entrada.
14	RELOJ	0	Reloj. Entero donde MSB = horas y LSB = minutos.
15	PBMENS	0	Devuelve el PBMensaje almacenado en el RCX.

Tabla 5.3

5.2.3. Ejecución del programa

- Guarda el proyecto
- Conecta un sensor de contacto en la entrada 1.
- Enciende el RCX.
- Ejecuta tu programa.
- Pulsa el botón Poll y un “0” deberá aparecer en el cuadro de texto.

[5] CW: giro en el sentido de las agujas del reloj. CCW: giro en contra del sentido de las agujas del reloj.

- Presiona y sujeta el pulsador del sensor y sin soltarlo pulsa otra vez Poll; un “1” deberá aparecer en el cuadro de texto..

5.3. Modos de sensores

El modo en el que las lecturas del sensor son devueltas puede ser cambiado. El método SetSensorMode instruye al RCX en qué modo quiere recibir los datos. La sintaxis general del método es SetSensorMode (Number, Mode, Slope)

Number (número) es un valor 0, 1 ó 2 y se refiere a los sensores SENSOR_1, SENSOR_2 y SENSOR_3 respectivamente.

Mode (modo) es el valor definido en la columna *Número* (o Constante) de la tabla 5.4.

Número	Constante	Modo del sensor	Descripción
0	RAW_MODO	Raw	Dato analógico sin tratar (0 – 1023)
1	BOOL_MODO	Booleano	Verdadero o Falso
2	TRANS_CONT_MODO	Transición	Se cuentan todas las transiciones (tanto las transiciones positivas como las negativas)
3	PERIOD_CONT_MODO	Contador periódico	Sólo cuenta periodos completos (una transición positiva + una negativa o viceversa)
4	PORCENT_MODO	Porcentaje	El valor del sensor se representa como el porcentaje de una escala completa..
5	CELSIUS_MODO	Celsius	Temperatura medida en Celsius
6	FAHRENHEIT_MODO	Fahrenheit	Temperatura medida en Fahrenheit
7	ANGULO_MODO	Ángulo	El dato de entrada se cuenta en pasos de ángulo

Tabla 5.4

Slope sólo se usa si el modo booleano ha sido elegido, y sino puede dársele el valor 0. Si el modo Booleano de operación es seleccionado, *Slope* indica cómo determinar el verdadero o falso en el SensorValue. Esto también afecta al modo en el que los contadores reaccionan ante los cambios en las entradas.

0: Medida absoluta (por debajo del 45% de la escala completa = verdadero, y por encima del 55% de la escala completa Falso). Por ejemplo, un sensor de contacto pulsado (medición de bajo voltaje) da un estado TRUE (verdadero).

1-31: medición dinámica. El número indica el tamaño del *slope* dinámico. Por ejemplo, el cambio necesario de bit-counts entre dos muestras, para producir un cambio en el estado booleano.

Nota: El método *SetSensorType* modifica automáticamente el modo para el sensor de contacto a Booleano, y el modo para el sensor de luz a

Porcentual. Esto quiere decir que siempre hay que invocar el método *SetSensorType* antes que el método *SetSensorMode*, ya que si no el método *SetSensorMode* no tendrá ningún efecto.

5.3.1. ComboBox y ListBox

Si quieres que sea posible desde el mismo programa pedir al RCX en qué modo deseas sean devueltas las lecturas de los sensores, puedes utilizar los ComboBox y ListBox.

Esto dos controles permiten tener una lista de elementos entre los que el usuario pueda elegir. Las diferencias entre los dos son pequeñas.

- Se puede escribir texto en un combo box durante la ejecución.
- Los dos tienen diferentes estilos, es decir, el ListBox no ofrece listas desplegables mientras que el ComboBox sí. Pueden ser utilizados en diferentes casos.

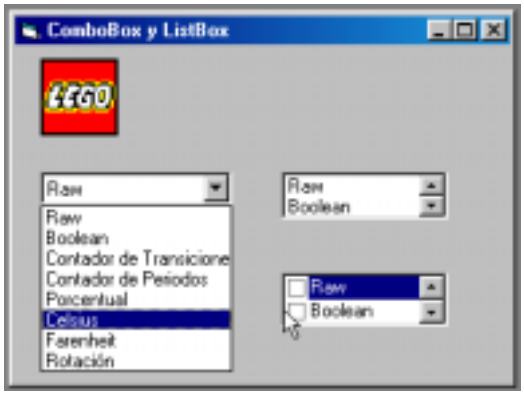


Figura 5.3:
A la izquierda control *ComboBox* y a la derecha L*istBox*

Comencemos a introducir los cambios necesarios:

- Haz un doble clic sobre el control *ComboBox* ( ) de la caja de herramientas.
- Dale el *Nombre* **cboModo**
- Para colocar valores en el *ComboBox* hazlo por medio de la propiedad *List*. Haz clic sobre la propiedad *List*, y a continuación haz clic sobre la flecha de la celda de la derecha.
- Escribe **Raw**
- Pulsa la combinación de teclas <**Ctrl.+Enter**> para llevar el cursor a la siguiente línea.
- Escribe **Boolean**.
- Pulsa la tecla <**Enter**> o haz clic en cualquier lugar exterior a la lista para completar la operación.

Figura 5.3
cambio de estilo en la ventana *Propiedades*

- Cambia el estilo (*Style*) a 2 – Dropdown List.

5.3.2. Código

```
Option Explicit
Dim iModo As Integer

Private Sub cmdSalir_Click()
    PBrickCtrl.CloseComm
    End
End Sub

Private Sub cmdPoll_Click()
    ‘Determinar cuál es el modo deseado
    If cboModo.ListIndex = 0 Then
        iModo = RAW_MODO
    ElseIf cboModo.ListIndex = 1 Then
        iModo = BOOL_MODO
    EndIf
    'Configura la entrada 1 como sensor de contacto
    PBrickCtrl.SetSensorType SENSOR_1, TIPO_SWITCH
    ‘Recupera el resultado como Booleano
    PBrickCtrl.SetSensorMode SENSOR_1, iModo, 0
    ‘La lectura del sensor al cuadro de texto
    txtPoll.Text = PBrickCtrl.Poll(SENVAL, SENSOR_1)
End Sub

Private Sub Form_Load()
    PBrickCtrl.InitComm ‘Inicializa el puerto serie del PC
    cmbModo.Text = cmbModo.List(0) ‘Visualiza el primer elemento
End Sub
```

5.3.3. Descripción del código

En el principio del código se declara la variable de tipo entero (Integer) iModo, la cual será utilizada para almacenar el valor del modo correspondiente al valor seleccionado en el *ComboBox*.

```
    ‘Determinar cuál es el modo deseado
    If cboModo.ListIndex = 0 Then
        iModo = RAW_MODO
    ElseIf cboModo.ListIndex = 1 Then
        iModo = BOOL_MODO
    EndIf
```

El primer valor del *ComboBox* tiene como valor 0, el siguiente 1... La propiedad *ListIndex* contiene el valor actual seleccionado en el *ComboBox*. Si el valor es cero, se asigna el valor

RAW_MODO a la variable iModo, y si el valor es uno, se asigna a la variable el valor BOOL_MODO.

```
PBrickCtrl.SetSensorMode SENSOR_1, iModo, 0
```

Por medio de esta instrucción el modo del sensor se configura con el valor almacenado en la variable iModo, el cual deriva del valor contenido en el *ComboBox*.

Dado que en el inicio hay que mostrar el valor por defecto, conviene introducir la siguiente instrucción en el procedimiento Form_Load. Dicha instrucción selecciona el primer elemento de la lista como valor por defecto cuando el programa arranca.

```
cboMode.Text = cboMode.List(0) ' Muestra el primer elemento.
```

- Guarda el proyecto
- Ejecuta el proyecto
- Selecciona ambas opciones y pulsa el botón Switch en cada una. Recuerda los valores de las lecturas.

5.3.4. Mejoras en el código

Gran parte del código que has escrito en el ejercicio anterior era innecesario. Podrás darte cuenta de ello observando los valores de índice del *ComboBox* y los valores numéricos de los diferentes modos, ya que coinciden siempre que se hayan introducido en el mismo orden.

- Añade lo siguiente a la propiedad *List* del *ComboBox* tal y como se ha descrito anteriormente

  Contador de transiciones
  Contador de periodos
  Porcentual
  Celsius
  Farenheit
  Angular

El *ListBox* debería verse tal y como se presenta en la figura 5.4 (Nota: La entrada Raw está presente pero fuera de la vista).

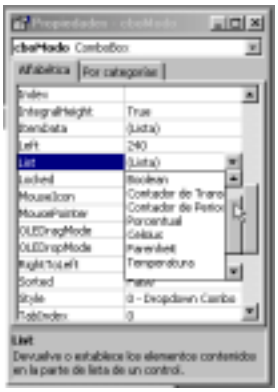


Figura 5.4
Tu lista debería contener los mismos elementos que aparecen aquí.

- Modifica el código de la manera siguiente:

```
Option Explicit

Private Sub cmdSalir_Click()
```

```
      PBrickCtrl.CloseComm
      End
End Sub

Private Sub cmdPoll_Click()
      ' configura la entrada 1 como sensor de contacto
      PBrickCtrl.SetSensorType SENSOR_1, TIPO_SWITCH
      ' devuelve el resultado en formato booleano
      PBrickCtrl.SetSensorMode SENSOR_1, cboModo.ListIndex, 0
      , da al cuadro de texto el valor del sensor 1
      txtPoll.Text = PBrickCtrl.Poll(SENVAL, SENSOR_1)
End Sub

Private Sub Form_Load()
      PBrickCtrl.InitComm 'Inicializa el puerto serie del PC.
      cboMode.Text = cboMode.List(0) ' Visualiza el primer elemento
End Sub
```

- Guarda otra vez el proyecto
- Ejecuta el proyecto
- Haz clic sobre *Poll*.

Figura 5.5
Si ejecutas el proyecto, podrás sondear diferentes sensores en el RCX.

Los tipos y modos de sensor pueden combinarse de diferentes formas, aunque las opciones angular, Celsius y Fahrenheit no son aplicables con el sensor de contacto.

5.4. Sensor de luz

Sería también interesante poder elegir el tipo de sensor durante la ejecución del programa (en este programa sólo podremos seleccionar el modo de sensor), de tal modo que sin hacer cambios en el programa podremos experimentar con diferentes combinaciones de modos y tipos de sensor.

- Coloca otro *ComboBox* en el formulario.

Tipo de control	Propiedad	Valor
ComboBox	Nombre List Style	CboTipo Ninguno Contacto Temperatura Luz Angular 2 – Drpdown List

Tabla 5.5

En esta ocasión, cuando introduzcas el código, utiliza el valor del *cboTipo.ListIndex* para configurar el tipo de sensor, y haz que el primer valor (Ninguno) sea la opción por defecto cuando el programa arranque.

- Guarda y ejecuta el programa otra vez.
- Una vez conectado el sensor, establece el modo de sensor y el tipo de sensor, y a continuación sondea los valores.

5.5. Organizador de bloques

El objeto de este nuevo proyecto es un "organizador de bloques" que sea capaz de diferenciar bloques LEGO de diferentes colores. Esto lo haremos a gracias a que los diferentes colores reflejan la luz con diferentes intensidades. En una primera fase sólo mediremos la intensidad de luz que refleja un color, y en una segunda fase completaremos el programa.

Las instrucciones para montar el organizador de bloques se encuentran en el apéndice A.

En las siguientes líneas se describe cómo empezar con el programa, pero quedará en tus manos finalizarlo.

- Guarda tu proyecto.
- Selecciona *Nuevo proyecto* en el menú *Archivo*.
- Selecciona el icono LEGO en la ventana *Nuevo proyecto*, y pulsa el botón *Aceptar*.
- Asegúrate que la ventana *Form1* del nuevo proyecto es la ventana seleccionada, y selecciona *Guarda Form1* como en el menú *Archivo*.
- Por medio del cuadro de diálogo que aparece, selecciona C:\VBLEGO\Cap05 como directorio para guardar el formulario.
- Escribe **Organizador** como nombre y pulsa el botón *Guardar*.
- Selecciona *Guardar proyecto como* en el menú *Archivo*.
- Primero se guardará el archivo .bas. Introduce el nombre **Organizador** y pulsa el botón *Guardar* (el destino debe ser también Cap05).
- Visual Basic te pedirá ahora el nombre del archivo .vbp. Escribe como nombre **Organizador** y pulsa el botón *Guardar*.

5.5.1. El control Tiempo (Timer)

Cada vez que un botón de comando es presionado, un procedimiento de evento asociado es ejecutado ("desencadenado"). Si deseas que una determinada acción suceda en automáticamente en intervalos regulares de tiempo, puedes usar un control *Timer*. Un control Timer permite que un procedimiento se ejecute en intervalos de tiempo previamente fijados. La propiedad *Interval* señala la longitud de los intervalos. Puede tener un valor entre 0 y 65535. Una unidad corresponde a un milisegundo (1 segundo es igual a 1000 milisegundos). Un control *Timer* es invisible durante la ejecución del programa, sólo es visible en el formulario durante la fase de diseño.

5.5.2. El control Shape

El control Shape es útil para dibujar varias formas geométricas:

- Rectángulos
- Cuadrados

- Círculos
- Elipses
- Cuadrados con vértices redondeados
- Rectángulos con vértices redondeados

Modificando la propiedad *Shape* del control *Shape* ( ) podremos dibujar la figura geométrica que deseemos. En este proyecto utilizaremos este control para expresar el color que ve el organizador. En el inicio se verá negro, pero cuando el organizador vea un bloque verde el rectángulo presente en el formulario se volverá verde.

- Construye el formulario de acuerdo con los datos de la tabla 5.6

Tipo de control	Propiedad	Valor
Form	Nombre Caption	FrmOrganizador Organizador de bloques
CommandButton	Nombre Caption	CmdSalir &Salir
Timer	Nombre Enabled Interval	TmrSondeo True 1000
TextBox	Name Text	TxtSondeo (vacío)
Label	Nombre Caption	LblSondeo Sensor de luz
Shape	Nombre BorderStyle FillStyle	ShpBloque 0 – Transparent 0 - Solid

Tabla 5.6

Figura 5.6
el formulario completo debería tener los mismos componentes que el de la figura.

- Escribe el siguiente código:

```
Option Explicit

Private Sub cmdSalir_Click()
      PBrickCtrl.CloseComm
      End
End Sub

Private Sub Form_Load()
```

```
      With PBrickCtrl
            .InitComm ‘Inicializa el puerto serie del PC.
            .SetSensorType SENSOR_1, TIPO_SWITCH ' Sensor 1 como sensor
      de contacto
            .SetSensorType SENSOR_3, TIPO_LUZ ' Sensor 3 como sensor de
      luz
            .SetSensorMode SENSOR_3, RAW_MODO, 0 ' Cambia de porcentual a
      Raw
      End With
End Sub
```

- Vuelve a la vista *Objeto*; haz un doble clic sobre el control *Timer* que has colocado sobre el formulario e introduce el siguiente código

```
Private Sub tmrSondeo_Timer()
      If PBrickCtrl.Poll(SENVAL, SENSOR_1) = 1 Then
            txtSondeo = PBrickCtrl.Poll(SENVAL, SENSOR_3)
            shpBloque.FillColor = QBColor(2)                ' verde
      Else
            shpBloque.FillColor = QBColor(0)                ' negro
      End If
End Sub
```

5.5.3. Ejecución del programa Organizador

- Guarda el proyecto
- Ejecuta el proyecto.

Al principio la figura geométrica está rellena en negro. Pulsa el sensor de contacto y verás que el valor del cuadro de texto cambia al valor Raw del sensor de luz y la figura geométrica se vuelve verde. Cuando sueltes el sensor de contacto el color se transformará otra vez en negro y en el cuadro de texto quedará el último valor medido.

5.5.4. Descripción del programa Organizador

Cuando se carga el programa los sensores son configurados uno como sensor de contacto (entrada 1) y el otro como sensor de luz (entrada 3). Fíjate en la instrucción *With*. Esta instrucción evita que tengas que escribir una y otra vez *PBrickCtrl* delante de todos los métodos que le corresponden.

```
With PBrickCtrl
```

Cuando ya no necesitemos el control escribiremos *End With.*

El procedimiento *tmrSondeo_Timer* se ejecuta cada 1000 ms (1 segundo). La primera línea de código

```
If PBrickCtrl.Poll(SENVAL, SENSOR_1) = 1 Then
```

comprueba si el sensor está presionado. Si lo está (es decir es igual a 1) la lectura del sensor de luz será asignada al cuadro de texto txtSondeo y el color de la figura geométrica se volverá verde. Si el sensor de contacto no está presionado se convierte en negro otra vez. Para ello modificaremos la propiedad FillColor del control shpBloque:

```
txtSondeo = PBrickCtrl.Poll(SENVAL, SENSOR_3)
shpBloque.FillColor = QBColor(2)                        ' verde
```

La función QBColor(2) devuelve el color correspondiente al valor de la tabla 5.7

Número	Color	Número	Color
0	Negro	8	Gris
1	Azul	9	Azul claro
2	Verde	10	Verde claro
3	Cyan	11	Cyan claro
4	Rojo	12	Rojo claro
5	Magenta	13	Magenta claro
6	Amarillo	14	Amarillo claro
7	Blanco	15	Blanco brillante

Tabla 5.7

5.5.5. Ejercicio

Modifica el programa Organizador de tal manera que sea capaz de diferenciar dos bloques de Lego de colores verde y azul colocados bajo el sensor. Las lecturas del sensor de luz varían en función del color sobre el que está situado. El control shape debería reflejar el color del bloque situado bajo el sensor de luz.

Todas las modificaciones deberían ser implementadas en el procedimiento tmrSondeo_Timer. Un esquema para ello puede ser el siguiente:

```
Private Sub tmrPoll_Timer()
      ' Declara un variable como entero para que guarde la lectura del
      sensor de luz
      Dim iLuzRaw As Integer
      If PBrickCtrl.Poll(SENVAL, SENSOR_1) = 1 Then
            iLuzRaw = PBrickCtrl.Poll(SENVAL, SENSOR_3)
            txtPoll = iLuzRaw
            ' Introduce el código que determina el color aquí.
      End If
End Sub
```

Nota: Puedes colocar una instrucción If … Then … Else dentro de otra, a lo que se llama anidar. Dado que *iLuzRaw* ha sido declarada en el interior del procedimiento *tmrPoll_Timer* y no en la sección previa de Declaraciones Generales, sólo puede ser utilizada en dicho procedimiento.

6 Variables en el RCX

6.1. Contenidos de este capítulo

Hasta ahora hemos utilizado algunas variables de Visual Basic. Dichas variables guardaban su valor en el ordenador. En este capítulo veremos cómo almacenar información en las variables del RCX.

1. Conocer el uso de variables en el RCX.
2. Uso de cadenas de texto y números.
3. Uso de mensajes de error.

6.2. Características de las variables en el RCX

En el RCX se pueden utilizar 32 variables globales. Estas variables pueden almacenar valores enteros comprendidos entre –32786 y 32767. Los valores de las variables pueden modificarse de diferentes maneras, entre otros por medio de sumas, restas, multiplicaciones y divisiones (en el caso de las divisiones el resultado se redondea al número entero más cercano). Los valores de las variables pueden ser sondeados desde el ordenador por medio del método Poll. Los nombres de las variables son números comprendidos entre 0 y 31.

6.3. Propuesta de proyecto

En este capítulo vamos a desarrollar un proyecto que tendrá como objetivo modificar y sondear el valor de las variables. Esto se podrá hacer por medio del formulario que ofrecerá el programa.

En un segundo paso haremos ciertas modificaciones en el programa para evitar errores que se puedan producir en la introducción de datos.

- Selecciona *Nuevo proyecto* en el menú *Archivo*
- Selecciona el icono Lego en la ventana *Nuevo Proyecto* y pulsa el botón *Aceptar*.
- Del mismo modo que has hecho en los capítulos anteriores, guarda todos tus nuevos archivos bajo el nombre Variables. Para ello crea el directorio C:\VBLego\Cap6 como destino para tus archivos.
- Crea un formulario a partir de los datos de la tabla 6.1

Tipo de control	Propiedad	Valor
Form	Nombre Caption	FrmVariable Manipulación de variables
CommandButton	Nombre Caption Font	cmdSet &Darle el valor Times New Roman 10
CommandButton	Nombre Caption Font	cmdPoll S&ondear el valor Times New Roman 10
CommandButton	Nombre Caption Font	cmdExit S&alir Times New Roman 10
TextBox	Nombre Alignement Font Text	TxtSetVar 2 – Center Times New Roman 10 (Dejarlo vacío)
TextBox	Nombre Alignement Font Text	TxtSetValor 2 – Center Times New Roman 10 (Dejarlo vacío)
TextBox	Nombre Alignement Font Text	TxtPollVar 2 – Center Times New Roman 10 (Dejarlo vacío)
TextBox	Nombre Alignement Font Text	TxtPollValor 2 – Center Times New Roman 10 (Dejarlo vacío)
Label	Nombre Alignement Caption Font	LblSet 2 – Center a la variable Times New Roman 10
Label	Nombre Alignement Caption Font	LblPoll 2 – Center de la variable Times New Roman 10

Tabla 6.1

Figura 6.1:
El formulario creado ha de tener un aspecto similar al de la figura

- Escribe el siguiente código:

```
'Todas las variables han de ser declaradas
Option Explicit

Private Sub cmdExit_Click()
      PBrickCtrl.CloseComm
      End
End Sub

Private Sub cmdPoll_Click()
      'Sondear el valor de la variable
      txtPollVar.Text= PBrickCtrl.Poll(VAR,Val(txtPollVar))
End Sub

Private Sub cmdSet_Click()
      'establecer el valor de la variable
      PBrickCtrl.SetVar Val(txtSetVar),CTE,Val(txtSetValor)
End Sub

Private Sub Form_Load()
      PBrickCtrl.InitComm
End Sub
```

- Guarda el proyecto.
- Ejecuta el proyecto.
- Enciende el RCX.
- Sondea el valor contenido en la variable nº 15.
- Asigna el valor 3333 a la variable nº 15.
- Sondea de nuevo la variable nº 15.

Se puede observar que la variable nº 15 contiene ahora el valor 3333.

6.3.1. Descripción del código

Con el objeto de asignar un valor a una variable, utilizaremos la instrucción SetVar:.

```
PBrickCtrl.SetVar Val(txtSetVar),CTE,Val(txtSetValor)
```

El contenido del cuadro de texto *TxtSetVar* es una cadena de texto (como en cualquier cuadro de texto), pero el método *SetVar* requiere un valor numérico. Por ello, deberemos convertir dicha cadena de texto en un valor numérico. Esto lo haremos utilizando la función *Val*(). Esta función se complementa con la función *Str*() que realiza la operación opuesta:

`Str(22334)="22334"`	Número	⇒	Cadena de texto
`Var("21")=21`	Cadena de texto	⇒	Número

La sintaxis de la instrucción SetVar es *SetVar (VarNo, Origen, Número)*. El primer argumento de la función PBrickCtrl.SetVar es el nombre de la variable que queremos modificar (entre 0 y 31). El segundo es el origen del valor a asignar, en este caso una constante, y el tercero el valor actual a asignar a la variable.

Para sondear una variable utilizaremos la siguiente instrucción:

```
PBrickCtrl.Poll(Origen,Número)
```

Origen y *Número* se utilizan para expresar qué hay que sondear.

En nuestro proyecto esto se refleja del siguiente modo:

```
TxtPollValorr.Text=PBrickCtrl.Poll(VAR,Val(txtPollVar))
```

Por medio de esta instrucción se comunica al RCX que queremos sondear una variable, y de qué variable se trata. En este caso deseamos sondear el valor numérico de la variable *txtPollVar*. Adviértase que en la línea de código anterior para asignar el valor devuelto por VAR,Val(txtPollVar) a PBrickCtrl.Poll, lo hemos tenido que encerrar entre paréntesis. Esto es para que el método Val(txtPollVar) se ejecute primero.

- Ejecuta el programa otra vez:
- Enciende el RCX.
- Asigna el valor 50 000 a la variable nº 23.

Veremos que sucede un error, ya que el número es demasiado grande (> 32767), Pulsa el botón *End* para cerrar el cuadro de diálogo *Error*.

- Asigna el valor 245 a la variable nº 50.

Ocurre otro error, ya que los nombres de variables son números comprendidos entre 0 y 31.

- Sal del programa

6.4. Cuadros de mensajes

A veces, cuando el programa necesita informar al usuario que algo acaba de ocurrir, se visualiza un mensaje en el monitor, generalmente con un botón *Aceptar*, para que el usuario pueda recibir el mensaje. En Visual Basic, puedes utilizar la instrucción *MsgBox* para crear tus propios cuadros de mensajes. Por ejemplo, si tuvieses un comando denominado *cmdMensaje*, podrías asociarlo con un cuadro de mensaje utilizando una instrucción similar a la siguiente:

```
Private Sub cmdMensaje_Click()
      MsgBox "Tu programa ha sido ejecutado con éxito", _
      vbExclamation, "Éxito"
End Sub
```

Este código generará un cuadro de mensaje igual al que recoge la figura 6.2, una vez se haya pulsado el botón *cmdMensaje*.

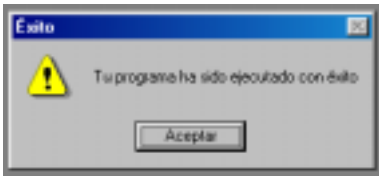


Figura 6.2
Este es el cuadro de mensaje que has creado

Antes de sondear una variable, hay que asegurarse de que el nombre de dicha variable sea un número comprendido entre 0 y 31. Para implementar este control, utilizaremos la estructura de control de Visual Basic If ... Then ... Else.

- Modifica el procedimiento *cmdPoll_Click* de acuerdo con el siguiente código, eligiendo tu mismo el mensaje deseado:

```
Private Sub cmdPoll_Click()
      If Val(txtPollVar)<0 Or Val(txtPollVar)>31 Then
            'Escribe aquí el mensaje deseado utilizando la instrucción_
      MsgBox
      Else
            TxtPollValor.Text=PBrickCtrl.Poll(VAR,Val(txtPollVar))
      End If
End Sub
```

- Guarda el programa.
- Enciende el RCX.
- Ejecuta el programa.
- Sondea la variable nº 41.

Un mensaje de error debería aparecer notificándote tu error.

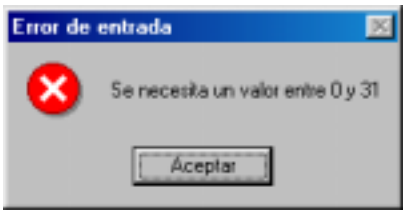


Figura 6.3
Un cuadro de mensaje de error.

6.4.1. Descripción del programa

En la primer línea de código del procedimiento *cmdPoll*_Click se encuentra la estructura de control If ... Then ... Else

```
If Val(txtPollVar)<0 Or Val(txtPollVar)>31 Then
```

Aquí se comprueban dos condiciones:

1. Si el valor numérico de *txtPollVar* es menor que cero.
2. Si el valor numérico de *txtPollVar* es mayor que treinta y uno.

Si una de las dos condiciones es verdadera, el valor está fuera de rango, por lo que hemos utilizado el operador *OR* para hacer cumplir esto.

La instrucción también podría escribirse de la siguiente manera utilizando el operador lógico *AND*:

```
If Val(txtPollVar)>=0 And Val(txtPollVar)<=31 Then
```

Esta instrucción comprueba si el valor es mayor o igual a 0 y (*AND*), al mismo tiempo, menor o igual a treinta y uno. Como es necesario que las dos condiciones sean verdaderas, utilizamos el operador lógico *AND*.

El primer método puede ser resumido como:

```
If Condición Then
Ha ocurrido un error
Else Todo correcto
```

Y el segundo método como:

```
If Condición Then
Todo correcto
Else Ha ocurrido un error
```

6.4.2. Ejercicio

Modifica el programa de tal manera que además de comprobar que el nombre de la variable esté comprendido entre 0 y 31, compruebe que el valor a asignar a una variable esté comprendido entre –32768 y 32767. Utiliza cuadros de mensaje para notificar al usuario su error.

6.5. Sondeo sistemático de variables

6.5.1. Estructuras de control iterativas

En muchas ocasiones, cuando se desarrolla un programa se necesita realizar una misma operación más de una vez. Por ejemplo, si has de construir un robot que haga repetidamente la misma cosa utilizaremos un bucle iterativo (iterativo significa repetitivo).

Un ejemplo de bucle iterativo es el bucle While ... Wend.

```
Dim i As Integer
i= 0
While i < 10
      Text1 = Str(i)+””
      i = i+1
Wend
```

En este ejemplo, inicialmente se asigna a la variable **i** (número entero) el valor 0. Cuando el programa encuentra la instrucción *While*, comprueba si la condición (**i** < 10) es verdadera o falda. En esta fase es verdadera, y la variable **i** será incrementada en una unidad. La instrucción *Wend* indica el final del código a repetir. En este punto el programa retrocede hasta la instrucción *While*, y comprueba otra vez si la condición es verdadera. Dado que todavía lo es (**i** = 1), incrementará su valor otra vez en una unidad. Este proceso se repite hasta que **i** = 10, momento en que la condición no se cumplirá. En consecuencia no se ejecutará más este bucle, y el programa continuará con la instrucción siguiente a *Wend* (el bucle se habrá ejecutado 10 veces).

Otro modo de generar un bucle de un modo en el que queda claro cuantas veces se va a realizar es utilizando la estructura de control *For ... Next*. Veámoslo por medio de un ejemplo:

```
Dim i As Integer
For i = 0 to 10
      Text1 = Str(i)+””
Next i
```

Una vez que la variable **i** ha sido declarada como número entero, el programa inicia el bucle *For ... Next*. Inicialmente se asigna el valor 0 a la variable **i**, y el bucle se ejecuta para valores desde 0 a 10. La línea sangrada escribe el valor de **i** y *Next i* incrementa repetidamente su valor en una unidad hasta que alcanza el 10. El bucle se habrá completado. Cuando el bucle haya finalizado el valor de **i** será 10.

Hay otros dos tipos de bucle en Visual Basic. Son el Do ... While ... Loop y Do ... Loop ... Until.

El bucle de la izquierda escribirá el número 9 cuando finalice, y el de la derecha también. Pero hay una diferencia entre los dos:

```
Dim i As Integer
i= 0
While i < 10
      Text1 = Str(i)+""
      i = i+1
Wend
```

```
Dim i As Integer
i= 0
Do
      Text1 = Str(i)+""
      i = i+1
Loop While i < 10
```

Tanto el bucle de la izquierda como el de la derecha escribirán al final el número 9 pero hay una diferencia entre los dos. La diferencia entre los dos tipos de bucle es que mientras que en el primero la comprobación de la condición se realiza al principio del bucle, en el de la derecha se hace al final del bucle. Las implicaciones de esto es mejor verlas por medio de un ejemplo. En este nuevo fragmento de código, asignaremos el valor 20 a **i** en lugar de 0 como en los casos anteriores.

```
Dim i As Integer
i= 20
While i < 10
      Text1 = Str(i)+""
      i = i+1
Wend
```

```
Dim i As Integer
i= 20
Do
      Text1 = Str(i)+""
      i = i+1
Loop While i < 10
```

Dado que en el bucle de la izquierda la comprobación se realiza al principio, no se producirá ninguna salida en el cuadro de texto. En el bucle de la derecha, el cuadro de texto visualiza el valor 20, ya que la comprobación se hace una vez que el bucle ha sido ejecutado una vez (el valor final de **i** en este segundo caso será 21, mientras que en el primero será 20). Dado que en el de la izquierda dado que no se ha ejecutado el bucle **i** no sufrirá cambios en su valor.

Ahora vamos a desarrollar un programa que lea los valores almacenados en las treinta y dos variables del RCX. Obtener el valor de las treinta y dos variables uno a uno sería muy largo y aburrido. Afortunadamente, podemos utilizar la estructura de control While ... Wend.

Añade los siguientes controles al formulario:

Tipo de control	Propiedad	Valor
CommandButton	Nombre Caption	cmdPollTodas Sondear &Todas
TextBox	Nombre Font Multiline ScrollBars Text	TxtTodasVar (elige una fuente de tu elección) True 2 - Vertical (Dejarlo vacío)[6]

Tabla 6.2

[6] Dado que la propiedad Multiline tiene el valor True (verdadero), el cuadro de texto se comportara de modo similar a la propiedad List del ComboBox.

Figura 6.4:
añade los componentes extras en el formulario.

- Escribe el siguiente código:

```
Private Sub cmdPollTodas_Click()
      Dim iContador As Integer
      Dim strTodasVar As String
      Dim strLineaActual As String
      Dim strLFCR As String
      StrLFCR = Chr(13)+Chr(10)
      iContador = 0
      While iContador <= 31
            strLineaActual = Str(iContador) + ":" + _
            Str(PBrickCtrl.Poll(VAR, iContador))
            strTodasVar = strTodasVar + strLFCR + strLineaActual
            iContador = iContador + 1
      Wend
      TxtTodasVar.Text = strTodasVar
End Sub
```

6.5.2. Descripción del programa

Para empezar se declaran varias variables de VB:

- iContador: se utiliza como contador en el bucle While ... Wend.
- StrTodasVar: contendrá todas las variables sondeadas (las leídas hasta el momento).
- StrLineaActual: Contiene el valor de la variable presente.
- StrLFCR: introduce un salto de línea para presentar los valores como una lista.

Verás que txtTodasVar muestra una larga cadena de texto que ocupará varias líneas. Para repartir la cadena de texto en varias líneas insertaremos entre las variables la cadena de texto **strLFCR** que colocará el valor de la siguiente variable en una nueva línea.

```
StrLFCR = Chr(13)+Chr(10)
```

Chr(13) es el carácter salto de línea y Chr(10) es el carácter linefeed.

El bucle *While ...Wend* ha de empezar con el valor 0 y finalizar con el 31. Esto se consigue asignando a la variable *iContador* el valor 0 antes de comenzar el bucle, y la condición se cumplirá siempre que su valor sea menor o igual a treinta y uno.

Veamos la instrucción:

```
strLineaActual = Str(iContador) + ":" +_
      Str(PBrickCtrl.Poll(VAR,iContador))
```

Por medio de esta línea de código vamos a hacer que la cadena *strLineaActual* contenga el número de la variable y su valor. Los elementos de esta suma de cadenas de texto son: el número de la variable, dos puntos y el valor de la variable actual.

Veamos ahora la línea

```
strTodasVar = strTodasVar + strLFCR + strLineaActual
```

Por medio de esta línea se añade a la cadena *strTodasVar* la cadena *strLFCR*, para que el valor de la siguiente variable se visualice en una nueva línea, y el contenido de *strLineaActual*. Al final del bucle esta variable contendrá los valores de todas las variables.

El procedimiento finaliza asignando el contenido de la variable *strTodasVar* al cuadro de texto *TxtTodasVar*.

6.5.3. Detección de cambios en una variable

En el quinto capítulo vimos cómo utilizar el control Timer para que el RCX leyese los valores de los sensores. El control Active-X Spirit.ocx puede realizar este sondeo de modo automático (controlando únicamente las modificaciones en las variables del RCX).

- Coloca un botón de comando en tu formulario y ponle el nombre *cmdAutoSondeo*, y da el valor *Auto&Sondeo* la propiedad *Caption*.
- Escribe el siguiente código:

```
Private Sub cmdAutoSondeo_Click()
      PBrickCtrl. SetEvent VAR,6,MS_200 'Configura el sondeo automático
End Sub
```

Este código establece el sondeo automático de la variable nº 6, en intervalos de tiempo de 200 milisegundos, pero además de ello, necesitamos algún elemento que no notifique el cambio: un cuadro de texto.

Asegúrate que estás en la vista *Código*.

- Elige PbrickCtrl en el ComboBox *Objeto* que se encuentra en la parte superior izquierda de la ventana *Código* (esto generará una plantilla que habrá de ser borrada).
- Elige *VariableChange* en la lista desplegable *Procedimiento* que se encuentra en la parte superior derecha de la misma ventana.
- En el interior de la plantilla generada por la selección anterior escribe el siguiente código:

```
Private Sub PBrickCtrl_VariableChange(ByVal Number As Integer, ByVal_
Value As Integer)
      'presenta el dato sondeado en un cuadro de mensaje
      MsgBox Str(Value), vbInformation, "La variable " + str(Number) + _
      " ha cambiado"
End Sub
```

Si ocurre algún cambio en la variable 6, el evento *PbrickCtrl_VariableChange* es enviado a la aplicación. A partir de ello cada uno puede decidir que hacer, en este caso, vamos a enviar un cuadro de mensaje al monitor para informar al usuario que el valor de dicha variable ha sido modificado.

- Ejecuta el programa.
- Enciende el RCX.
- Pulsa el botón AutoSondeo[7].
- Ahora da el valor 1234 a la variable 6

Un cuadro de mensaje aparecerá en el monitor informando sobre dicho cambio.

6.6. Ejercicio

Coloca un nuevo botón en el formulario que resetee todas las variables asignándoles el valor 0.

[7] Si la variable (en este caso la 6) no tiene el valor 0, el cuadro de mensaje aparecerá nada más pulsar el botón AutoSondeo. Si sucede esto, pulsa el botón Aceptar y sigue.

7 Robots autonomos

Los programas que hemos hecho hasta ahora, han sido directamente ejecutados desde el ordenador en tiempo real (por ejemplo, el control de motores por medio de un formulario). A este método se le denomina *Control Inmediato*. A continuación veremos otro método que permitirá transferir los programas desde el ordenador al RCX. Gracias a este método, el RCX podrá ejecutar los programas aunque se encuentre lejos de la torre de infrarrojos. Cuando se ejecutan tareas transferidas desde el ordenador sin recibir instrucciones adicionales desde él, decimos que el robot funciona de modo autónomo.

En este capítulo y en los siguientes utilizaremos la base de robot que se describe en el apéndice A. Dependiendo de la función que haya de cumplir, le añadiremos uno u otro sensor. Antes de seguir adelante móntalo con el sensor de contacto.

7.1. Objetivos de este capítulo

1. Programación de un robot autónomo: estructura de un programa.
2. Transferencia de programas al RCX. Control de errores en la transferencia.
3. Estructuras de control: estructuras iterativas y condicionales.
4. Sistemas motor y sensorial de un robot: toma de decisiones a partir de la información recogida por el aparato sensorial..

7.2. Estructura de un programa

El RCX tiene cinco slots de programa. En cada slot de programa puede almacenar ocho subrutinas y 10 tareas. Las tareas son fragmentos de código que pueden ser ejecutadas simultáneamente. Por ejemplo, en este capítulo desarrollaremos un robot que se desplazará de un lado a otro evitando obstáculos.

7.2.1. Propuesta de proyecto

El objetivo de este primer proyecto es muy simple: montar y programar un robot que se desplace durante tres segundos hacia adelante, para a continuación desplazarse durante otros tres hacia atrás, hasta quedar en la posición inicial.

7.2.2. Edición del programa

- Inicia Visual Basic.
- Selecciona el icono Lego en la ventana *Nuevo Proyecto* y pulsa el botón *Aceptar*
- Guarda todos tus nuevos archivos bajo el nombre **Bajar** del mismo modo que has hecho en los capítulos anteriores. Para ello crea el directorio C:\VBLego\Cap7 como destino para tus archivos.
- Crea un formulario a partir de los datos de la tabla 7.1.

Figura 7.1:
este será el sencillo formulario de nuestro programa

Tipo de control	Propiedad	Valor
Form	Nombre Caption	frmBajar Programa Bajar
CommandButton	Nombre Caption	cmdBajaPrograma &Baja el programa
CommandButton	Nombre Caption	cmdExit &Salir

Tabla 7.1

El código será el siguiente:

```
Private Sub cmdBajaPrograma _Click()

    With PBrickCtrl

        .SelectPrgm 2 'o SelectPrgm SLOT_3 utilizando la constante

        .BeginOfTask 0

            .On MOTOR_A+MOTOR_B
            .SetFwd MOTOR_A+MOTOR_B
            .Wait 2, SEG_3
            .SetRwd MOTOR_A+MOTOR_B
            .Wait 2, SEG_3
            .Off MOTOR_A+MOTOR_B

        .EndOfTask

    End With

End Sub

Private Sub Form_Load()
      PBrickCtrl.InitComm
End Sub

Private Sub cmdExit_Click()
      PBrickCtrl.CloseComm
      End
End Sub
```

7.2.3. Ejecución del proyecto

- Guarda el proyecto

- Enciende el RCX.
- Ejecuta el programa
- Pulsa el botón *Baja el Programa* (o <Ctrl+B>).
- En el display del RCX aparecerá el número 3, señalando que el programa elegido es el 3.
- Pulsa el botón **Run** del RCX

El robot deberá moverse durante tres segundos hacia adelante, retrocederá durante otro tres y se parará en el punto de salida.

7.2.4. Descripción del programa

Analicemos paso a paso este programa. Para no utilizar la referencia PBrickCtrl una y otra vez utilizaremos al principio del programa la instrucción *With ... End With*.

```
With PBrickCtrl
```

A partir de ella no tendremos que escribir otra vez PBrickCtrl.

.SelectPrgm 2 : se utiliza para elegir el slot de programa a utilizar. El argumento de la instrucción es un número comprendido entre 0 y 4 (corresponden a los slot de programa 1 a 5). En lugar de números pueden utilizarse las constantes definidas en el módulo RCXdatos.bas.

.BeginOfTask 0 : esta instrucción da inicio a la tarea principal. En el RCX los programas se componen de tareas. Todos los programas necesitan de una tarea principal: la tarea 0 (si utilizas el módulo RCXdatos.bas puedes sustituir el número 0 por la constante PRINCIPAL). Esta será la tarea que se ejecutará cuando pulsemos el botón **Run** del RCX. La estructura de las tareas siempre será la siguiente:

```
.BeginOfTask 0
      .instrucción 1
      .instrucción 2
      ...
      .instrucción n
.EndOfTask
```

El código comprendido entre *.BeginOfTask* y *.EndOfTask* describe lo que pasará cuando pulsemos el botón **Run**. El código comprendido entre estas dos instrucciones será el que se transferirá al RCX, y nada más.

En el programa actual, los motores 1 y 3 se pondrán en marcha con toda la potencia posible, el programa detendrá su ejecución durante tres segundos (Wait), cambiará el sentido de giro de los motores y tras otros tres segundos parará los motores, con lo que finalizará la ejecución del programa.

7.3. Detección de errores

Vamos a añadir un sistema de detección de errores en nuestro programa. Hasta ahora hemos supuesto de un modo optimista que todos los comandos que transmitíamos al RCX eran recibidos correctamente por él. Veamos qué ocurriría si el programa no fuese correctamente transferido al RCX.

El evento *DownloadDone* es enviado por el control Active X tan pronto como la transferencia al RCX ha finalizado, o cuando un error detiene prematuramente la transferencia. El evento no aparece en el formulario (The event is of the form)

```
PBrickCtrl_DownloadDone(ByVal ErrorCode As Integer, ByVal DownloadNo As_
Integer)
```

Si *ErrorCode* es igual a cero, quiere decir que la transferencia ha sido realizada de modo correcto. Pero si su valor es uno, quiere decir que la transferencia ha sido fallida, y *DownloadNo* señala a qué número de tarea o subrutina corresponde el error.

7.3.1. Edición del código

- Selecciona PbrickCtrl en el ComboBox *Objeto* que se encuentra en la parte superior izquierda de la ventana *Código* (esto generará una plantilla que habrá de ser borrada).
- Selecciona *DownloadDone* en la lista desplegable *Procedimiento*.
- En el interior de la plantilla generada por la selección anterior escribe el siguiente código:

```
Private Sub PBrickCtrl_DownloadDone(ByVal ErrorCode As Integer, ByVal _
DownloadNo As Integer)
      If ErrorCode=0 Then           'Éxito en la transferencia
            PBrickCtrl.PlaySystemSound SWEEP_DOWN_SOUND
            MsgBox "Transferencia correcta",vbInformation,"Status"
      Else                          'Errores en la transferencia
            MsgBox "Transferencia fallida",vbCritical,"Status"
      End If
End Sub
```

7.3.2. Ejecución del programa

- Guarda el proyecto.
- Ejecuta el proyecto
- Apaga el RCX.
- Pulsa el botón *Transfiere programa*.

Tras unos segundos, un cuadro de mensaje aparecerá con un mensaje de error para informar que la transferencia ha sido fallida.

- Haz clic en Aceptar para cerrar el cuadro de mensaje.
- Enciende ahora el RCX.
- Pulsa otra vez el botón *Transfiere programa.*

Si la transferencia es correcta, el RCX generará el sonido SWEEP_DOWN y un cuadro de mensaje aparecerá para informar que la transferencia se ha desarrollado de modo correcto.

7.4. Estructuras de control de flujo

Las estructuras de control de flujo pueden ser utilizadas en este modo de trabajo del mismo modo que se utilizan en Visual Basic. Hay tres tipos básicos:

.Loop

.While

.If ... Else

7.4.1. Loop

La estructura de control Loop repite todos los comandos que engloba un número determinado de veces.

```
PBrickCtrl.Loop CTE, 4
      PBrickCtrl.PlaySystemSound BEEP_SOUND
PBrickCtrl.EndLoop
```

La primera parte de la estructura (Loop) contiene el número de veces que la estructura ha de ser repetida. En este caso el origen es una constante, y el valor de la constante es cuatro. Adviértase que la estructura Loop es menos ambigua que cualquier otra como While ... Wend o For .... Next, ya que refleja de modo explícito el número de veces que se ha de repetir. Sin embargo, hay un punto negativo ya que en otros tipos de bucles iterativos la variable utilizada para controlar las iteraciones puede ser utilizada en el mismo cuerpo del bucle. En nuestros anteriores ejemplos imprimíamos el valor actual de la variable que controlaba la iteración. Con la estructura Loop perdemos esta capacidad con el objeto de hacerla más sencilla.

El método *EndLoop* hace disminuir en una unidad el valor del contador (en este caso, cuatro al principio) y comprueba si el valor resultante es igual a cero. Si es así, el bucle finaliza y el siguiente comando es ejecutado, en caso contrario, los comandos comprendidos en el bucle se ejecutarán de nuevo.

El código del ejemplo hará sonar BEEP_SOUND cuatro veces.

Un caso particular es cuando se utiliza la instrucción Loop CTE, SIEMPRE (Loop 2,0) al principio del bucle. Esto significará que el bucle se repetirá indefinidamente. Esto sucede si el origen de datos es una constante. En cambio, si el origen es una variable (por ejemplo: Loop 0,5) y el valor de la variable en el momento de ejecutar la instrucción *Loop* es cero, el bucle no se ejecutará, y se seguirá con la instrucción que siga al *EndLoop*.

7.4.2. While

La estructura de control While ... EndWhile es similar a la estructura de control Do While ... Loop que hemos visto en Visual Basic. Su sintaxis es la siguiente:

While (Origen1, Número1, RelOp, Origen2, Número2)

Los dos primeros valores se refieren al primer valor a comparar, y los dos últimos al segundo valor de la comparación. El parámetro RelOp describe cómo han de ser comparados los valores. Hay cuatro diferentes modos de comparación.

Número	Constante	Descripción
0	MAYQ	Mayor que
1	MENQ	Menor que
2	IG	Igual
3	DIF	No igual a

Tabla 7.2

```
With PBrickCtrl
      .SetVar 6, CTE,1
      .While SENVAL,SENSOR_1,IG,VAR,6
            .PlaySystemSound BEEP_SOUND
            .Wait CTE,MS_500
      .EndWhile
End With
```

El valor 1 es asignado a la variable nº 6 del RCX. El primer valor a comparar en la estructura de control While es la lectura del sensor 1, y el segundo el número contenido en la variable nº 6 (en este caso la unidad).

Por lo tanto, esta estructura establece que mientras la lectura del sensor 1 sea igual a 1 genere un sonido Beep cada medio segundo.

7.4.3. If ... Else

La estructura de control If ... [Else] ... EndIf compara dos valores del mismo modo que la estructura de control While.

If (Origen1, Número1, RelOp, Origen2, Número2)

Si la condición es verdadera, los comandos que se encuentran a continuación de la instrucción *If* se ejecutaran, y si es falsa los que vengan después de Else. La instrucción *Else* (sino) es opcional, o sea, simplemente puede finalizar la estructura *If* sin más alternativas.

```
With PBrickCtrl
      .SetVar 6, CTE, 800
      .If SENVAL, SENSOR_3, MENQ, VAR, 6
            .On MOTOR_A
      .Else
            .On MOTOR_B
      .EndIf
End With
```

7.5. Un robot que evita obstáculos

7.5.1. Propuesta de proyecto

Vamos a construir ahora un robot que sea capaz de evitar obstáculos. Construye el robot de acuerdo con el modelo del apéndice A. Añade a la base común el módulo que incluye un sensor de contacto.

El programa puede hacerse de modos diferentes. Algunos ofrecerán mejores resultados que otros. El algoritmo para desarrollar el de aquí veremos es el siguiente:

```
Si el robot toca algo hará lo siguiente:
      - retroceder durante un segundo
      - girar a un lado
Mientras no toque nada avanzará
```

7.5.2. Edición del programa

- Crea un nuevo botón de comando en el formulario y ponle como nombre **cmdContacto**.
- Da a la propiedad *Caption* el valor **Programa &Contacto**.
- Escribe el siguiente código:

```
Private Sub cmdContacto_Click()
      With PBrickCtrl
            .SelectPrgm SLOT_4      'Slot de programa nº 4
            .BeginOfTask PRINCIPAL
                  .SetSensorType SENSOR_1, TIPO_SWITCH
                  .SetPower MOTOR_A + MOTOR_C, CTE, 3
```

```
                .Loop CTE, SIEMPRE
                      .If SENVAL, SENSOR_1, IG, CTE, 1 ' Si sensor =_
                      pulsado
                            .SetRwd MOTOR_A + MOTOR_C
                            .Wait CTE,SEG_1              ' retrocede
                            .Off MOTOR_C
                            .Wait CTE,SEG_1              ' gira
                            .Off MOTOR_A
                      .Else
                            .SetFwd MOTOR_A + MOTOR_C
                            .On MOTOR_A + MOTOR_C        ' avanza
                      .EndIf
                .EndLoop
          .EndOfTask
    End With
End Sub
```

- Guarda el proyecto
- Enciende el RCX.
- Ejecuta el proyecto
- Transfiere el programa al RCX pulsando el botón **Programa Contacto**.
- Coloca el robot en el suelo o en otra superficie adecuada, y ejecuta el programa.

Podrás observar que cuando el robot choca con algo, retrocede e intenta rodear el obstáculo.

7.5.3. Descripción del programa

Para empezar se selecciona el slot nº 4 del RCX como destino del programa. Las dos primeras líneas de la tarea principal (las dos líneas siguientes a BeginfTask PRINCIPAL) configuran adecuadamente el sensor de contacto y establecen cuál ha de ser la potencia que han de desarrollar los motores. La instrucción

```
Loop CTE, SIEMPRE
```

hace que el programa genere un bucle sin fin.En este bucle lo que se repite una y otra vez es lo siguiente:

```
Si el sensor de contacto está presionado
      Invertir la marcha durante un segundo, y a continuación girar
      durante otro segundo
Sino
      Avanzar.
```

El tipo de superficie sobre el que se mueve el robot tiene consecuencia en su comportamiento. Por ejemplo, no es lo mismo que se mueva sobre una alfombra o sobre baldosas. Por ello, quizás tengas que modificar el tiempo durante el que el robot tiene que girar o retroceder (argumento de la instrucción *Wait*).

7.6. Ejercicio

1. En el programa anterior cambia el modo del sensor a modo *Raw*, y haz las modificaciones que sean necesarias en la condición *If*. Por medio de la experimentación pedes ver qué lectura da el sensor de contacto cuando está presionado y cuál cuando está sin presionar (o cuando está medio pulsado).
2. En el mismo programa, puedes observar que mientras el sensor de contacto no esta presionado, el programa está ordenando una y otra vez al RCX que ponga en

marcha los motores, aunque ya esté avanzando. Optimiza el código para que ponga en marcha el robot en el inicio de la tarea, y sólo ordene otra vez que avance tras una maniobra de giro.

3. En la parte delantera del robot en lugar de colocar un sensor colocaremos dos para saber si los choques suceden en la parte izquierda o derecha. Si choca con la derecha el robot girará a la izquierda y viceversa.

8 Siguiendo una línea

En este capítulo vamos a construir y programar robots con un sistema sensorial basado en sensores de luz. Las diferentes condiciones de luz en las que se desarrolle la experimentación con dichos robots, puede requerir que sea necesario ajustar ciertos valores utilizados en los programas.

8.1. Contenidos de este capítulo

1. Profundización en la programación del RCX.
2. Uso del sensor de luz como sensor de proximidad: emisión y detección de luz infrarroja.

8.2. Un robot que sigue una línea

8.2.1. Propuesta de proyecto

Vamos a empezar con un proyecto clásico: un robot que sea capaz de seguir una línea negra trazada en el suelo. Para empezar a trabajar con este tipo de robots, lo más adecuado es hacerlo sobre circuitos descritos por medio de líneas negras sobre fondo blanco (líneas de negro bien saturado, no el que conseguimos con un rotulador). El poster que acompaña al kit MindStorms representa una línea ovalada negra y vamos a programar el robot para que la siga.

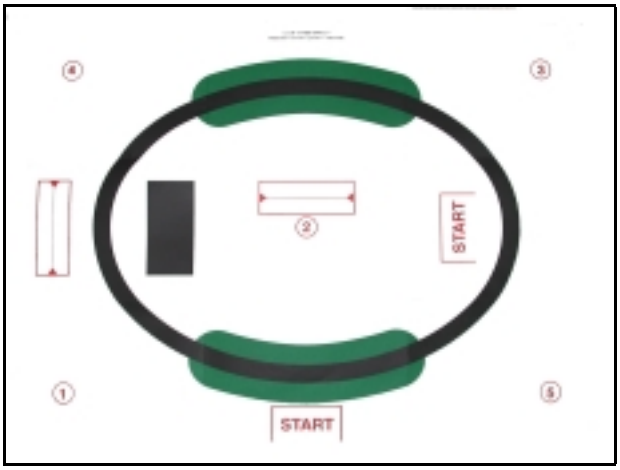


Figura 8.1
El "Test Pad" de LEGO MindStorms

En este proyecto pueden utilizarse uno o dos sensores de luz. Cualquiera podría pensar que es más fácil hacerlo con dos sensores de luz, ya que así podríamos saber por qué lado de la línea se desvía. Sin embargo, con un solo sensor puede ser suficiente convirtiéndose en una interesante propuesta que permite desarrollar estrategias diversas.

El robot que utilizaremos será el mismo que el del capítulo anterior, pero en este caso con el módulo correspondiente al sensor de luz (apéndice A). El sensor de luz ha de estar conectado en la entrada 3, ya que en caso contrario el programa no funcionará de modo correcto.

Para empezar decidimos que el robot ha de girar siguiendo el óvalo en el sentido de las agujas del reloj. Así, cuando el robot se salga de la línea lo hará siempre por la izquierda, y lo único que tendrá que hacer para volver a la línea será girar a la derecha. Una vez visto esto, el programa no ofrece mayores dificultades.

> El robot avanzará mientras el sensor de luz le informe que se encuentra sobre la línea negra.
>
> Al salir de la línea detendrá el motor de la derecha, de tal modo que durante cierto tiempo (por medio de la experimentación se podrá determinar durante cuanto tiempo ha de detenerse) sólo el motor de la izquierda avanzará (esto es, se moverá hacia la derecha).

8.2.2. Edición del código

Una vez que hayas montado el robot convertiremos el algoritmo anterior en un programa::

- Abre el proyecto creado en el capítulo anterior.
- Crea un nuevo botón de comando en el formulario y ponle como nombre **cmdLínea**.
- Da a la propiedad *Caption* el valor **Programa &Línea**.
- Escribe el siguiente código:

```
Private Sub cmdLínea_Click()
      With PBrickCtrl
            .SetSensorType SENSOR_3, TIPO_LUZ
      End With
End Sub
```

Este código sirve para determinar el umbral de las lecturas de luz que indicará que se ha salido de la línea negra. El sensor 3 es un sensor de luz configurado en modo porcentual.

- Guarda y ejecuta el programa.
- Enciende el RCX.
- Pulsa el botón **Programa Línea** para configurar el sensor 3 del RCX.
- Por medio del botón **View** del RCX, selecciona el sensor 3 para ver su lectura (la flecha presente en el LCD debe apuntar al sensor 3, tal y como se puede ver en la figura 8.2).
- Mueve el sensor de luz sobre las zonas blanca y negra del poster que acompaña el kit MindStorms para obtener las lecturas correspondientes a las zonas blanca y negra. Mueve el sensor de modo similar al que se moverá cuando el robot esté acabado.

Figura 8.2
En el LCD del RCX la flecha señala el sensor 3.

- Escribe el siguiente código:

```
Private Sub cmdLínea_Click()

      Const ARC_TIEMPO = 5          ' Nombre de la variable 5
      Const UMBRAL_LUZ = 6          ' Nombre de la variable 6

      With PBrickCtrl
            .SelectPrgm SLOT_4
            .BeginOfTask PRINCIPAL
                  .SetVar ARC_TIEMPO, CTE, MS_50
                  .SetVar UMBRAL_LUZ, CTE, XXXX 'Introduce aquí tu valor
                  .SetSensorType SENSOR_3, TIPO_LUZ
                  .SetPower MOTOR_A + MOTOR_C, CTE, 6
                  .On MOTOR_A + MOTOR_C
                  .Loop CTE, SIEMPRE
                        .While SENVAL, SENSOR_3, MAYORQ, VAR, UMBRAL_LUZ
                              .Off MOTOR_C
                              .Wait VAR, ARC_TIEMPO
                        .EndWhile
                        .On MOTOR_C
                  .EndLoop
            .EndOfTask
      End With
End Sub
```

- Guarda y ejecuta el proyecto.
- Transfiere el programa Luz al RCX.
- Coloca el RCX sobre el poster con el sensor de luz sobre la línea negra de modo que el sentido de giro del robot sobre la trayectoria ovalada sea en el sentido de las agujas del reloj.
- Pulsa **Run** en el RCX

El RCX debe desplazarse a lo largo de la línea negra del poster.

8.2.3. Descripción del programa

El programa comienza nombrando las dos variables que utiliza con el objeto de hacer el programa más legible. Da el nombre ARC_TIEMPO a la variable nº 5, ya que esta variable define el tiempo que se detiene el motor C para hacer volver al robot a la línea negra (durante ese tiempo el robot describirá una trayectoria en arco). Y da el nombre UMBRAL_LUZ al umbral que define el límite entre la luz reflejada por la línea negra y el fondo blanco.

En el inicio del código que transferiremos al RCX se asigna el valor necesario a las dos variables. La razón de utilizar variables en lugar de valores constantes es facilitar el trabajo, ya que de este modo, si hay que hacer algún cambio en el valor de ARC_TIEMPO o UMBRAL_LUZ únicamente habremos de hacerlo en una línea y no cada vez que se utiliza dicho valor.

Una vez configurados los sensores y la potencia de los motores los motores se ponen en marcha y la tarea entra en un bucle sin fin (*Loop*). Si en este bucle el sensor de luz detecta que el robot ha salido de la línea, el programa detiene el motor C, espera un periodo de tiempo (definido al principio del programa por medio de la variable ARC_TIEMPO) y comprueba otra vez si el robot está fuera de la línea negra. Esto lo hará hasta que el robot encuentre de nuevo la línea negra (bucle *While*), momento en el que pondrá en marcha otra vez el motor C. El bucle Loop continuará ejecutándose hasta que pierda otra vez la línea negra, con lo que se repetirá el procedimiento anterior.

8.2.4. Ejercicios

1. Por ahora el robot sólo puede seguir la línea en el sentido de las agujas del reloj. Intenta modificar el código de tal manera que pueda seguir la línea en cualquier dirección.

   Sugerencia: busca la línea a un lado, y si no la encuentra es que está en el otro lado. Otra estrategia puede ser seguir el borde de la línea (la lectura del sensor será un valor intermedio).

2. Los cambios que hemos hecho en el ejercicio anterior han sido en el software. En este ejercicio se trata de modificar tanto el software como el hardware. Ahora el robot habrá de utilizar dos sensores de luz.

3. Programa el robot de tal modo que se mueva en el interior del óvalo de un lado a otro pero sin salir de él.

8.3. Robot proximidad

Cuando se utiliza un sensor de contacto para evitar obstáculos, el método es más bien primitivo, y por lo tanto no siempre funciona. Sería mejor solución que el robot pudiese detectar que va a chocar antes de hacerlo.

Para ello es necesario conocer bien las características del sensor de luz y del RCX. A primera vista el método para hacer esto puede no parecer obvio, pero si profundizamos en el funcionamiento del sensor de luz, podemos ver que tiene cierta sensibilidad ante la luz infrarroja. Utilizando este nuevo conocimiento, es posible construir un robot que sea capaz de detectar obstáculos. Será necesario disponer de una fuente de luz infrarroja, y para ello disponemos del RCX, ya que sabemos que las transmisiones entre el ordenador y el RCX se hacen por medio de luz infrarroja. Por lo tanto, podemos emitir luz infrarroja desde el RCX en intervalos regulares utilizando el método *SendPBMessage*. El sensor de luz puede aprovechar las fluctuaciones en las lecturas para saber si se encuentra cerca de un objeto.

- Modifica el robot del modo en que se explica en el apéndice A.
- En el formulario anterior, coloca un nuevo botón de comando y ponle como nombre **cmdProxi**, y sustituye el contenido de la propiedad *Caption* por **P&rograma Proximidad**.
- Escribe el siguiente código:

```
Private Sub cmdProxi_Click()

      Const ULTIMA_LECTURA = 10
      Const FLUCTUACION = 11

      With PBrickCtrl
            .SelectPrgm SLOT_5

            .BeginOfTask PRINCIPAL
                  .SetVar FLUCTUACION, CTE, 100
                  .StartTask 1
                  .StartTask 2
            .EndOfTask

            .BeginOfTask TAREA_1
                  .Loop CTE,SIEMPRE
                        .SendPBMessage CTE, 0
                        .Wait CTE,MS_10
```

```
            .EndLoop
      .EndOfTask

      .BeginOfTask TAREA_2
            .SetSensorType 2, TIPO_LUZ
            .SetSensorMode 2, RAW_MODO, 0
            .SetFwd MOTOR_A + MOTOR_C
            .On MOTOR_A + MOTOR_C
            .Loop CTE, SIEMPRE
                  .SetVar ULTIMA_LECTURA, SENVAL, SENSOR_3
                  .SumVar ULTIMA_LECTURA, VAR, FLUCTUACION
                  .If SENVAL, SENSOR_3, MAYORQ, VAR, ULTIMA_LECTURA
                        ' Obstáculo detectado
                        ' mueve el robot para evitar el obstáculo
                        ' y avanza otra vez
                  .EndIf
            .EndLoop
      .EndOfTask

   End With
End Sub
```

- Guarda y ejecuta el proyecto.
- Transfiere el programa **Proximidad** al robot.
- Ejecuta el programa.

Cuando el robot se acerque a un obstáculo, retrocederá e intentará evitarlo.

8.3.1. Descripción del programa

Al principio del procedimiento dos constantes han sido declaradas con el objeto de hacer más legible el código.

```
Const ULTIMA_LECTURA = 10

Const FLUCTUACION = 11
```

En la tarea principal se asigna un valor a la variable FLUCTUACION. Este valor se elige en función de la sensibilidad que queramos dar al robot (cuanto menor sea mayor será la sensibilidad).

Este es la primera vez a lo largo de este manual que vamos a utilizar más de una tarea en un programa. Dado que el botón **Run** únicamente inicia la tarea principal, el resto de las tareas han de ser iniciadas manualmente.

Se ejecutarán simultáneamente dos tareas: la primera (1) se ocupará de enviar periódicamente señales infrarrojas, mientras que la segunda (2) interpretará las lecturas del sensor y tomará las decisiones que sean necesarias para evitar los obstáculos.

```
.BeginOfTask TAREA_1
      .Loop CTE, SIEMPRE
            .SendPBMessage CTE, 0
            .Wait CTE, MS_10
      .EndLoop
.EndOfTask
```

La función de esta tarea es transmitir una señal infrarroja cada 10 ms, utilizando para ello el método *SendPBMessage*.

La segunda tarea comienza estableciendo el tipo y modo de sensor (modo Raw[8]). También pone en marcha los dos motores con lo que el robot comienza a avanzar. A continuación la tarea inicia un bucle sin fin:

```
.Loop CTE, SIEMPRE
      .SetVar ULTIMA_LECTURA, SENVAL, SENSOR_3
      .SumVar ULTIMA_LECTURA, VAR, FLUCTUACION
      .If SENVAL, SENSOR_3, MAYORQ, VAR, ULTIMA_LECTURA
            ' Obstáculo detectado
            ' mueve el robot para evitar el obstáculo
            ' y avanza otra vez
      .EndIf
.EndLoop
```

La lectura actual del sensor es asignada a la variable ULTIMA_LECTURA. A continuación se suma el valor de la variable FLUCTUACION (con valor 100 en este ejemplo) a la variable ULTIMA_LECTURA. Si en algún momento la lectura del sensor supera el valor de la variable ULTIMA_LECTURA, eso significará que hay algo cerca del sensor de luz (algo que refleja la luz infrarroja).

8.3.2. Ejercicio

Completa el programa proximidad para que el robot evite los obstáculos.

[8] El modo Raw (0 … 1023) tiene mayor sensibilidad que el modo porcentual (0 ... 100), es decir, mayor precisión.

9 Registro de datos

9.1. Contenidos de este capítulo

En el capítulo sexto has aprendido a utilizar las 32 variables del RCX. Quizás sean suficientes en la mayoría de los proyectos, pero habrá ocasiones en las que necesites almacenar más datos. Por ejemplo, en el caso en que queramos registrar los cambios de intensidad de luz que lee el sensor de luz de un robot móvil. Para ello el RCX nos ofrece el "datalog", es decir, el registro de datos.

Los datos así registrados no los vamos a poder utilizar del mismo modo que el contenido de las variables. Estos datos los transferiremos al ordenador, y si así lo deseamos los presentaremos por medio de un gráfico. Para almacenar estos datos en el ordenador necesitaremos utilizar variables, pero dado que serán numerosos datos utilizaremos otro tipo de variables: las matrices.

1. Variables en Visual Basic: matrices.
2. Registro de datos en el RCX.
3. Menús en los formularios.
4. Caja de Herramientas: uso de un control gráfico para presentar en el formulario los datos registrados en el RCX de modo gráfico.

9.2. Matrices (array)

Una matriz no es mucho más que una lista de variables. Con matrices, puedes almacenar una gran cantidad de datos similares, por ejemplo, las intensidades de luz leídas en diferentes momentos. Si para guardar este tipo de datos utilizásemos variables normales (las que no son matrices) a cada dato le correspondería una variable de diferente nombre y sería complicado controlar tal conjunto de datos.

Una matriz es un conjunto de datos del mismo nombre. Veámoslo por medio de un ejemplo: se trata de almacenar los resultados que un grupo de estudiantes ha obtenido en un examen.

```
Dim Resultado As Integer            'nota del alumno/a
```

Esta instrucción declara una variable *Resultado* como entero. Esta variable podría referirse a los resultados en los exámenes de un alumno. Si hubiese más de un estudiante en clase, habría que declarar una variable por cada estudiante, lo que supondría un largo y tedioso trabajo. En estos casos son útiles las variables de tipo matricial.

Los diferentes elementos de la matriz se diferencian entre sí por un subíndice. En lugar de los diferentes nombres de variables (Result01, Result02, Result03...) los datos correspondientes reciben el mismo nombre de variable y se diferencian por los subíndices. Por ejemplo:

Resultado01	Resultado(1)
Resultado02	Resultado(2)
…	…

El número entre paréntesis es el subíndice de la matriz. Los subíndices no son nunca parte del nombre de las matrices; siempre se encuentran entre paréntesis y sólo sirven para diferenciar un elemento de la matriz de otro. Si tienes que calcular la media de un conjunto de resultados de un examen utilizando únicamente variables, sería necesario teclear todos los nombres de variables uno por uno, mientras que con matrices, puedes utilizar un bucle Loop ... Next para sustituir los nombres de las variables.

Para un grupo de 40 estudiantes:

Con variables:

```
iTotal = Resultado01 + Resultado02 + Resultado03 + … + Resultado40
iMedia = Total / 40
```

Con matrices:

```
For iContador = 1 To 40
      iTotal = iTotal + Resultado(iContador)
Next iContador
iMedia = Total / 40
```

Tal y como se puede ver, incluso con sólo 40 estudiantes, el código se reduce utilizando matrices.

9.2.1. Declaración de matrices

```
Dim MiMatriz(10) As Integer
```

Esta matriz contiene 11 elementos [de MiMatriz(0) a MiMatriz(10)]. 0 es conocido como *límite inferior* y 10 como *límite superior*.

El límite inferior también puede ser especificado en el momento de la declaración.

```
Dim MiMatriz(10 To 20) As Integer
```

Este es el modo de declarar una matriz de 11 elementos cuyo límite inferior es 10, y el límite superior 20.

9.2.2. Matrices multidimensionales

Una matriz multidimensional es una matriz con más de un subíndice. Una matriz de una dimensión es una lista de valores, mientras que las matrices multidimensionales se asemejan a tablas de valores. La tabla más usada es la tabla bidimensional (una matriz con dos subíndices). Volviendo al ejemplo de los estudiantes, si cada estudiante tiene más de un examen (por ejemplo 6), se puede utilizar una matriz para almacenar los resultados.

```
Dim MiMultiMatriz(1 To 40, 1 To 6) As Integer
```

Esto es parecido a declarar una tabla de 40 filas y 6 columnas, donde cada fila es un alumno y cada columna una asignatura.

9.3. Registro de datos (datalog)

El datalog es un área situada en el interior del RCX, la cual permite almacenar lecturas que provienen de:

- Temporizadores.
- Variables.
- Sensores
- Reloj (tiempo)

Para utilizar el datalog, primero has de establecer el tamaño del área datalog que deseas usar. Esto se hace utilizando el método *SetDatalog*(*Tamaño*). El tamaño corresponde al número de elementos que se desea almacenar (cada elemento ocupa 3 bytes).

Para almacenar un valor en el datalog en cualquier momento de un programa, se utiliza *DatalogNext(Origen,Número).*

Origen		Número	
Constante		Constante	
0	VAR	0 – 31	
1	TEMPOR	0 – 3	TEMPOR_1, TEMPOR_2, TEMPOR_3, TEMPOR_4,
9	SENVAL	0 – 2	SENSOR_1, SENSOR_2, SENSOR_3
14	WATCH	0	

Tabla 9.1

Una vez finalizada la ejecución del programa, se puede transferir la información desde el RCX al ordenador, utilizando el método *UploadDatalog(From,Tamaño).*

9.4. Proyecto

- Crea un nuevo proyecto Lego.
- Guárdalo con el nombre **Registro de datos**
- Crea un formulario a partir de los datos de la siguiente tabla:

Tipo de control	Propiedad	Valor
Form	Nombre Caption	Frm_Datalog Registro de datos
CommandButton	Nombre Caption	CmdSetDLTamaño &Configura Datalog
CommandButton	Nombre Caption	CmdLimpiaDL &Vacía Datalog
CommandButton	Nombre Caption	CmdCargaDL &Recupera Datalog
CommandButton	Nombre Caption	CmdTransfiere &Transferir programa
CommandButton	Nombre Caption	CmdSalir &Salir
TextBox	Nombre Text	TxtDLtamaño 5
Label	Nombre Caption	LblDatalog (Dejarlo vacío)
List Box	Nombre	LstDatalog

Tabla 9.2

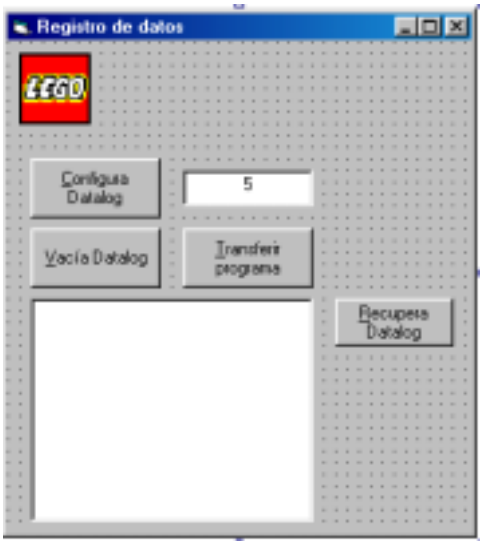


Figura 9.1:
Formulario para el registro de datos

9.4.1. Código

Introduce el siguiente código:

```
Private Sub cmdCargaDL_Click()
    Dim Mat As Variant
    Dim iContador As Integer
    ' Bajar el Datalog a la matriz Mat
    Mat = PBrickCtrl.UploadDatalog(0, Val(TxtDLtamaño.Text) + 1)
    If IsArray(Mat) Then
      For iContador = LBound(Mat, 2) To UBound(Mat, 2)
        LstDatalog.AddItem "Tipo: " + Str(Mat(0, iContador)) + " Nº :" +_
            Str(Mat(1, iContador)) + "Valor :" + Str(Mat(2, iContador))
      Next iContador
    Else
            MsgBox "La matriz no es válida"
    End If
End Sub

Private Sub cmdSalir_Click()
    PBrickCtrl.CloseComm
    End
End Sub

Private Sub cmdTransfiere_Click()
    With PBrickCtrl
        .SelectPrgm SLOT_4
        .BeginOfTask PRINCIPAL
            .SetSensorType SENSOR_2, TIPO_LUZ
            .SetVar 10, CTE, 1234
            .Loop CTE, 3
                .DatalogNext TEMPOR, TEMPOR_4   ‘ Valor del temporizador
                .Wait CTE, SEG_1
            .EndLoop
            .DatalogNext SENVAL, SENSOR_2       ‘ lectura del sensor
            .DatalogNext VAR, 10                ‘ valor de la variable 10
            .DatalogNext TEMPOR, TEMPOR_4       ‘ Valor del temporizador
        .EndOfTask
    End With
End Sub
```

```
Private Sub cmdLimpiaDL_Click()
    PBrickCtrl.SetDatalog 0                    'Vacía el Datalog
End Sub

Private Sub cmdSetDLTamaño_Click()
    If PBrickCtrl.SetDatalog(Val(TxtDLtamaño.Text)) Then
        DatalogLbl.Caption = "El tamaño del Datalog es " + TxtDLtamaño
    Else
        DatalogLbl.Caption = "No hay suficiente memoria disponible"
    End If
End Sub

Private Sub Form_Load()
    PBrickCtrl.InitComm
End Sub
```

9.4.2. Ejecución del programa

- Guarda el programa
- Ejecuta el programa
- Conecta el sensor de luz en la entrada 2
- Enciende el RCX
- Escribe el valor 7 en el cuadro de texto y pulsa el botón *Configura Datalog* (no escribas un número mayor que 50 en el cuadro de texto).
- Pulsa el botón *Transferir Programa* para bajarlo al RCX.
- Pulsa el botón **Run** del RCX
- Cuando finalice la ejecución del programa, pulsa el botón *Recupera Datalog* (no escribas un número superior a 50 en el textbox cuando pulses el botón *Recupera Datalog*).

En el ListBox aparecerán 7 entradas. La entrada del datalog correspondiente a 0 guarda el tamaño del datalog. El resto de las entradas son los valores almacenados por medio del método *DatalogNext*. Los datos numerados con 1, 2 y 3 son los valores del temporizador registrados, el siguiente la lectura del sensor, el contenido de la variable 10 y para acabar, otra vez el valor del temporizador.

Puedes observar que al pulsar el botón *Configura Datalog* aparece un cuadrante en la pantalla LCD del RCX. Al pulsar el botón **Run** del RCX aparecerán nuevos cuadrantes hasta completar el círculo. Para borrar el contenido del datalog pulsa el botón *Vacía Datalog*.

9.4.3. Descripción del programa

cmdSetDLtamaño: Este procedimiento establece el tamaño del datalog a partir del valor contenido en el textbox *txtDLtamaño*. El tamaño máximo varía, pero se encuentra alrededor de 2000. Si no hay suficiente memoria disponible, el método *SetDatalog*(Val(txtDLtamaño.Text)) falla y un mensaje de error aparece en el label lblDatalog.

cmdTransfiereDL: La función transferida al RCX, guarda el valor de TEMPOR_4 cada segundo durante tres veces, y a continuación la lectura del sensor es guardada en el Datalog. Seguidamente, el valor de una variable es introducido en el datalog, para Acabar introduciendo otra vez la lectura del temporizador.

cmdCargaDL: Este procedimiento carga el datalog desde el RCX en una matriz.

```
Mat = PBrickCtrl.UploadDatalog(0, Val(TxtDLtamaño.Text) + 1)
```

Hay que empezar por el primer elemento del datalog (0) y continuar hasta alcanzar el final del datalog. La razón de añadir el valor 1 es que el primer elemento del datalog contiene el tamaño actual, por ejemplo, han sido añadidas a la lista seis entradas, aunque hay siete elementos para cargar desde la lista.

La matriz devuelta es una matriz de dos dimensiones. La matriz contendrá tres filas y txtDLtamaño + 1 columnas.

Si la matriz es correcta:

```
For iContador = LBound(Mat, 2) To UBound(Mat, 2)
      LstDatalog.AddItem "Tipo: " + Str(Mat(0, iContador)) + " Nº :" +_
      Str(Mat(1, iContador)) + "Valor :" + Str(Mat(2, iContador))
Next iContador
```

El límite inferior de la matriz está definido (por ejemplo, la posición del primer elemento) y el límite superior también (por ejemplo, la posición del último elemento). Y para cada elemento comprendido entre estos dos valores hay una entrada.

Tipo		Número	Lectura
0	ALD	0 – 31	Lecturas devueltas
1	TEMPOR	0 – 3	
9	SENBALIO	0 – 2	
14	ERLOJU	0	

Tabla 9.3

El datalog se vacía estableciendo su tamaño a cero. El cuadrante desaparece entonces del display LCD del RCX.

```
Private Sub cmdLimpiaDL_Click()
    PBrickCtrl.SetDatalog 0              'vacía el Datalog
End Sub
```

9.5. Programa gráfico

En el programa anterior el conjunto de datos recuperados desde el datalog se ha presentado como una lista. Sin embargo, generalmente presentar los datos por medio de gráficos es mucho más expresivo. El objetivo de la segunda propuesta de este capítulo es introducir algunas modificaciones en el programa proximidad del capítulo anterior: las lecturas del sensor de luz se registrarán y, a continuación, se presentarán en modo gráfico.

En este programa utilizaremos menús, procedimientos y gráficos (PictureBox).

- Inicia un nuevo proyecto Lego.
- Guarda todos los archivos bajo el nombre **Grafico**.

En este proyecto es necesario todo el espacio posible en el formulario para el gráfico. Para conseguirlo, incorporaremos menús al programa.

9.5.1. Menús

En los anteriores programas hemos utilizado botones en el formulario. Ahora, los sustituiremos por menús. En la figura 9.2 puede verse el resultado al que queremos llegar.

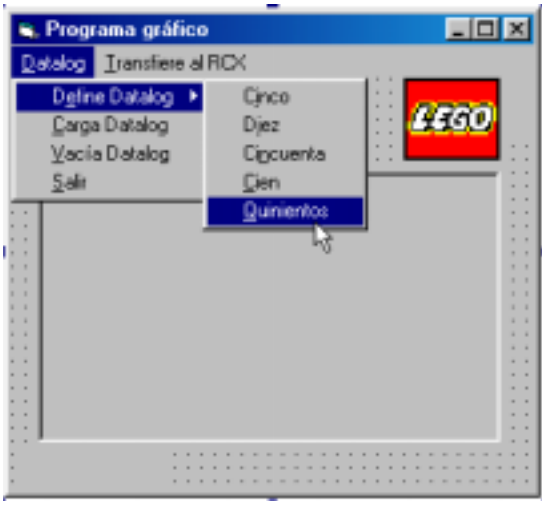


Figura 9.2:
menús en el formulario

- Crea el formulario frmGráfico de acuerdo con la tabla 9.4.

Tipo de control	Propiedad	valor
Form	Nombre Caption	frm_Gráfico Programa Gráfico

Tabla 9.4

- Selecciona el formulario
- Selecciona el *Editor de menús* en el menú *Herramientas*.

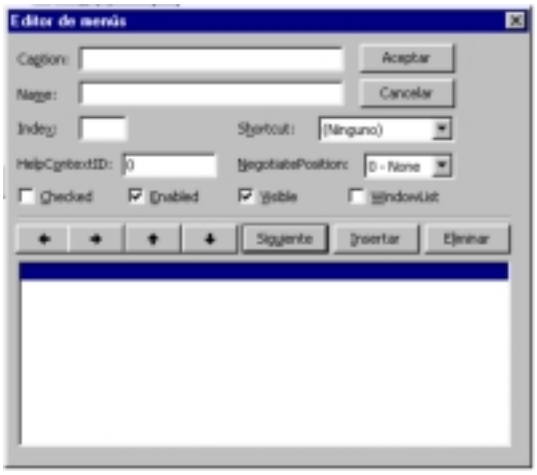


Figura 9.3:
editor de menús

- En el textbox *Caption* escribe **&Datalog**.
- En el textbox *Name* teclea **mnudatalog.**

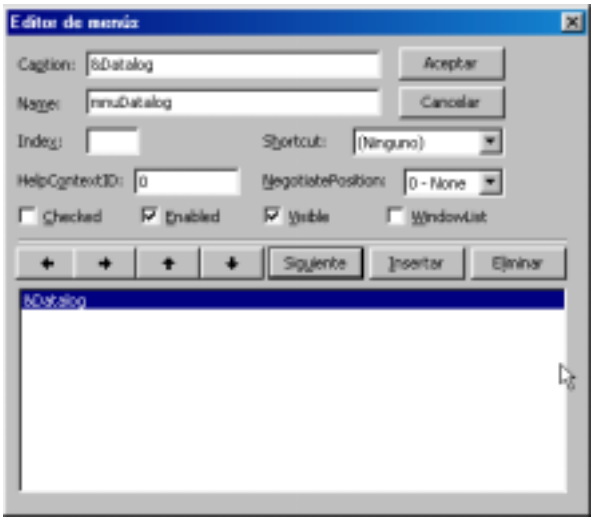


Figura 9.4:
entrada Datalog en el editor de menús

- Pulsa el botón *Siguiente* del *Editor de menús*, y la fila siguiente aparecerá resaltada en color azul.
- En el textbox *Caption* escribe **D&efine Datalog**.
- En el textbox *Name* teclea **mnuSet.**

Dado que *Define Datalog* es un elemento del menú *Datalog*, es necesario sangrarlo.

- Pulsa el botón correspondiente a la flecha hacia la derecha del *Editor de menús.*

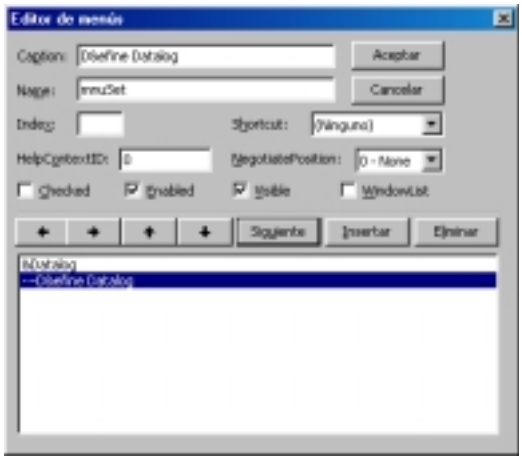


Figura 9.5
el elemento *Define Datalog* sangrado

- Pulsa el botón *Siguiente*.
- En el textbox *Caption* escribe **&Carga Datalog.**
- En el textbox *Name* teclea **mnuCarga**.
- Pulsa el botón *Siguiente*.
- En el textbox *Caption* escribe **&Vacía Datalog**.
- En el textbox *Name* teclea **mnuBorra**.
- Pulsa el botón *Siguiente*.

- En el textbox *Caption* escribe **&Salir**.
- En el textbox *Name* teclea **mnuSalir**.

El menú Datalog está completo. A continuación hay que crear el menú que transfiera el programa al RCX..

- Pulsa el botón *Siguiente* en el *Editor de menús*.
- En el textbox *Caption* escribe **&Transfiere al RCX**.
- En el textbox *Name* teclea **mnuDownload**

Dado que este es el nombre de un nuevo menú, y no un elemento de un menú, hay que eliminar el sangrado.

- Pulsa el botón correspondiente a la flecha hacia la izquierda del *Editor de menús* para eliminar el sangrado.
- Pulsa el botón *Siguiente*.
- En el textbox *Caption* escribe **&Programa proximidad**.
- En el textbox *Name* teclea **mnuProxi**
- Pulsa el botón correspondiente a la flecha hacia la derecha del *Editor de menús* para sangrar este elemento.

Ya está completo el menú, que deberá presentar el aspecto que se puede ver en la figura 9.6.

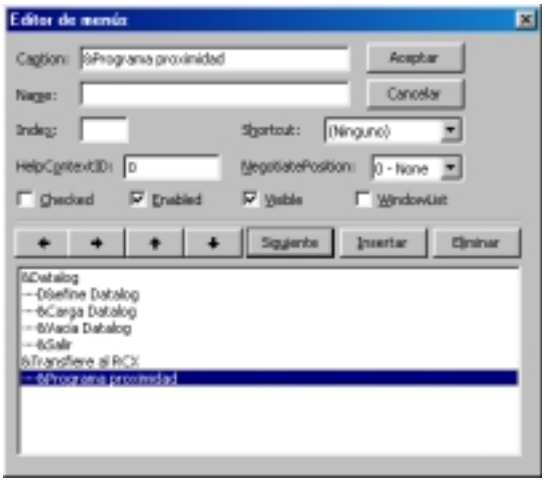


Figura 9.6:
el editor de menús una vez finalizada la edición

- Pulsar *Aceptar* en el *Editor de menús*.
- Guardar el proyecto.

El formulario frmGráfico tendrá la apariencia que se puede ver en la figura 9.7. Si pulsas sobre *Datalog* o *Transfiere al RCX* podrás ver como se despliegan los menús.

Figura 9.7
el formulario acabado

- Ejecutar el programa.

Se puede pulsar y seleccionar las opciones de los menús, aunque evidentemente, no sucederá nada, ya que todavía no les hemos asociado código.

- Pulsa la el icono X de la esquina superior derecha del programa *Gráfico* para salir del programa.

9.5.2. Creación de un submenú

Te habrás dado cuenta que una de las opciones del menú *Datalog* es *Define Datalog*. Sabemos que el método *SetDatalog* requiere un parámetro que notifique al control ActiveX el tamaño del datalog. Vamos a crear un submenú para ello.

- Selecciona el *Editor de menús* del menú *Herramientas*.
- Selecciona el elemento *Carga Datalog*, y pulsa el botón *Insertar*.
- En el textbox *Caption* escribe **C&inco**.
- En el textbox *Name* teclea **mnuCinco.**
- Pulsa el botón de la flecha a derechas para sangrarlo más.
- Selecciona de nuevo el elemento *Carga Datalog*, y pulsa el botón *Insertar*.
- En el textbox *Caption* escribe **Di&ez**.
- En el textbox *Name* teclea **mnuDiez.**
- Pulsa el botón de la flecha a derechas para sangrarlo más.
- Inserta del mismo modo los siguientes elementos:
- En el textbox *Caption* escribe **Di&ez**.
- En el textbox *Name* teclea **mnuDiez.**
- Pulsa el botón de la flecha a derechas para sangrarlo más.
- Inserta del mismo modo los siguientes elementos:

Caption	Name
Ci&ncuenta	MnuCincuenta
&Cien	MnuCien
&Quinientos	MnuQuinientos

Tabla 9.5

- Guardar el proyecto.

9.5.3. Control gráfico

- Seleccionar el control Picture Box de la caja de herramientas y colócalo en el formulario.
- Cambia la propiedad *Nombre* por **picGrafico**. Ahora el formulario frmGráfico tendrá el aspecto de la figura 9.8.

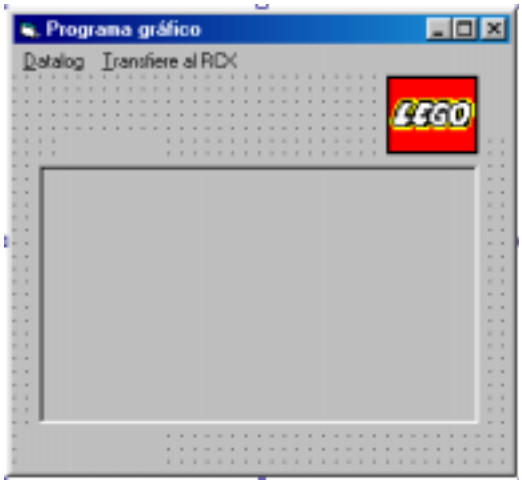


Figura 9.8
el formulario acabado

9.5.4. Código del programa gráfico

- Introduce el siguiente código en el programa:

```
'Todas las variables han de ser declaradas
Option Explicit

Private Sub Form_Load()
    PBrickCtrl.InitComm
End Sub
```

Ahora has de introducir el código correspondiente al elemento *Salir* del menú *Datalog*.

- En modo Diseño, en la vista *Objeto* haz clic sobre el menú Datalog y selecciona el elemento *Salir*.

Esto es parecido al doble clic sobre un botón comando, el shell para el procedimiento *mnuSalir_Click* aparece ahora en la ventana *Código*

- Introduce el siguiente código:

```
Private Sub mnuSalir_Click()
      PBrickCtrl.CloseComm
      End
End Sub
```

- Guarda y ejecuta el programa.
- Selecciona el elemento *Salir* del menú *Datalog*.

El programa finaliza.

9.5.5. Procedimientos

En el submenú de *Define Datalog* hay varias opciones para definir el tamaño del datalog que va a ser creado. Cuando se configura el datalog, hay que verificar que el datalog ha sido creado (por ejemplo, si hay suficiente espacio disponible). En lugar de escribir código para que verifique esto en cada opción, se puede crear un procedimiento que lo haga automáticamente.

- Con la ventana *Código* abierta seleccionar *Agregar Procedimiento* en el menú *Herramientas*.
- En el textbox *Name* escribir **SetDatalog**.

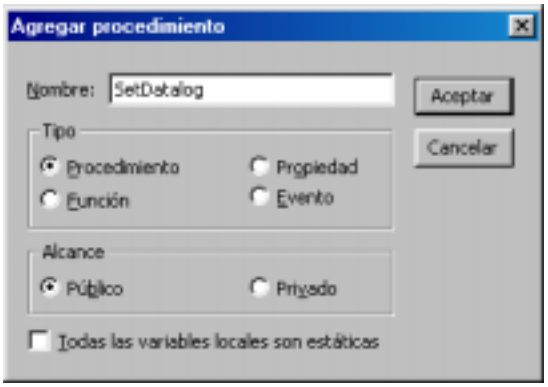


Figura 9.9
cuadro de diálogo *Agregar Procedimiento*

- Pulsa el botón *Aceptar*.

Aparece un shell para la el nuevo procedimiento:

```
Public Sub SetDatalog()

End Sub
```

- Ahora hay que modificar la primera línea del procedimiento *SetDatalog*:

```
Public Sub SetDatalog(Tamaño As Integer)

End Sub
```

- Escribe el siguiente código:

```
Public Sub SetDatalog(Tamaño As Integer)
    If PBrickCtrl.SetDatalog(Tamaño) Then
        MsgBox " Tamaño del Datalog: " + Str(Tamaño), vbInformation
    Else
        MsgBox " No hay suficiente memoria disponible", vbCritical
    End If
End Sub
```

- En la vista *Objeto*, selecciónala opción *Define Datalog* ⇒ *Cinco* del menú *Datalog*

El siguiente shell de procedimiento aparecerá:

```
Private Sub mnuCinco_Click()

End Sub
```

- Escribe el siguiente código:

```
Private Sub mnuCinco_Click()
    SetDatalog 5
End Sub
```

Esta instrucción ejecuta el procedimiento *SetDatalog* que acabamos de crear pasándole el número cinco como parámetro. Cuando el procedimiento es ejecutado, la variable *Tamaño* toma el valor 5.

- Repetir el procedimiento anterior para el resto de los elementos del submenú *Define Datalog*.

Vamos a añadir el código a la opción *Programa Proximidad*.

- En la vista *Objeto* seleccionar *Programa Proximidad* en el menú *Transfiere al RCX*.
- Introducir el siguiente código. Adviértase que este código es casi el mismo que el del capítulo anterior, excepto que tiene una tarea más

```
Private Sub mnuProxy_Click()
    Const ULTIMA_LECTURA = 10
    Const MARGEN = 11

    With PBrickCtrl
        .SelectPrgm SLOT_5

        .BeginOfTask PRINCIPAL
            .SetVar MARGEN,CTE,100
            .StartTask TAREA_1
            .StartTask TAREA_2
        .EndOfTask

        ' la tarea envía mensajes continuamente, dos por segundo

        .BeginOfTask TAREA_1
            .Loop CTE,SIEMPRE
                .SendPBMessage CTE, 0
                .Wait CTE, MS_50
        .EndOfTask

        'la siguiente tarea tiene el control del vehículo

        .BeginOfTask TAREA_2
            .SetSensorType SENSOR_2, TIPO_LUZ
            .SetSensorMode SENSOR_2, MODO_RAW, 0
            .SetFwd MOTOR_A + MOTOR_C
            .On MOTOR_A + MOTOR_C

            .StartTask TAREA _3
            .Loop CTE, SIEMPRE
                .SetVar ULTIMA_LECTURA, SENVAL, SENSOR_2
                .SumVar ULTIMA_LECTURA, VAR, MARGEN
                .If SENBALIO, SENSOR_2, MAYQ, VAR, ULTIMA_LECTURA
                    .SetRwd MOTOR_A + MOTOR_C
                    .Wait CTE, SEG_1
                    .Off MOTOR_C
                    .Wait CTE, SEG_1
                    .SetFwd MOTOR_A + MOTOR_C
                    .On MOTOR_C
                .EndIf
            .EndLoop
```

```
        .EndOfTask

        ' guarda diez lecturas por segundo en el datalog

        .BeginOfTask TAREA_3
            .Loop CTE, 100
                .DatalogNext SENVAL, SENSOR_2
                .Wait CTE, MS_100
            .EndLoop
            .Off MOTOR_A + MOTOR_C
            .StopAllTasks
        .EndOfTask

    End With
End Sub
```

Añade el código al elemento *Carga Datalog* del menú *Datalog*.

- En la vista *Objeto* seleccionar *Carga Datalog* del menú *Datalog*
- Introducir el siguiente código:

```
Private Sub mnuCarga_Click()
    Dim iLotes, i, iCuenta As Integer
    Dim Mat As Variant
    Dim iX, iSup, iInf As Integer
    Dim iLimInfX, iLimSupX, iLimInfY, iLimSupY As Integer

    Mat = PBrickCtrl.UploadDatalog(0, 1)
    iSup = Mat(2, 0)

    'define los límites del gráfico
    iLimInfX = 0: iLimSupX = iSup   ‘x entre 0 y el número de elementos
    iLimInfY = 500: iLimSupY = 850 ‘y con valores entre 500 y 850
    iX = 0                          'x comenzará en la coordenada 0

    picGrafico.Cls
    picGrafico.Scale (iLimInfX, iLimSupY)-(iLimSupX, iLimInfY)
    picGrafico.ForeColor = QBColor(4)

    iLotes = Int(iSup / 50)         ‘parte entera del cociente

    For iCuenta = 0 To iLotes
        iInf = iCuenta * 50
        If iSup <= 50 Then
            Mat = PBrickCtrl.UploadDatalog(iInf,iSup)
        Else
            Mat = PBrickCtrl.UploadDatalog(iInf,50)
        End If
        iSup = iSup - 50
        If IsArray(Mat) Then
            For i = LBound(Mat, 2) To UBound(Mat,2)
                iX = iX + 1
                picGrafico.Line -(iX,Mat(2,i))
            Next i
        Else
            MsgBox "No es una matriz válida"
        End If
    Next iCuenta

End Sub
```

- Para el elemento *Vacía Datalog* introduce el siguiente código:

```
Private Sub mnuBorra_Click()
    SetDatalog 0 'borra el Datalog
End Sub
```

Has acabado con la edición del programa, así que ya lo puedes ejecutar. Utilizaremos el mismo robot que en capítulo anterior: el robot proximidad.

- Guarda el proyecto.
- Ejecuta el proyecto.
- Seleccionar en el menú *Datalog* la opción *Define Datalog* ⇒ *Cien*.
- Selecciona *Programa Proximidad* en el menú *Transfiere al RCX* para transferir el programa al robot.
- Pulsar el botón **Run** en el RCX.
- Cuando el programa haya finalizado, seleccionar Carga Datalog en el menú Datalog (para ello debes colocar el RCX cerca de la torre de infrarrojos para que no se genere ningún problema de comunicaciones).

Un gráfico semejante al de la figura 9.10 debe aparecer en el PictureBox del formulario. En este ejemplo el robot ha encontrado dos obstáculos. La velocidad con el que el robot se acerca y se aleja de los obstáculos determina lo amplios que son los mínimos.

Figura 9.10
las lecturas del sensor de luz representadas de modo gráfico.

- Salir del programa. Si se quiere limpiar el *Datalog*, selecciona *Vacía Datalog* en el menú *Datalog* antes de salir del programa.

9.5.6. Descripción del funcionamiento del programa

Una vez el sensor de luz comienza a tomar lecturas en el Programa Proximidad, la tarea 3 es iniciada.

```
.BeginOfTask TAREA_3
      .Loop CTE, 100
          .DatalogNext SENVAL, SENSOR_2
          .Wait CTE, MS_100
      .EndLoop
      .Off MOTOR_A + MOTOR_C
      .StopAllTasks
.EndOfTask
```

La tarea 3 ejecuta el bucle que contiene 100 veces. En cada ciclo del bucle, guarda la lectura del sensor de luz en el Datalog. Una vez finalizado realizados los 100 ciclos del bucle, todas las tareas son detenidas (es decir, se detiene el programa).

El procedimiento *mnuUpload_Click* coloca el gráfico en el picture box. El primer paso para ello será transferir el contenido del datalog al ordenador.

La primera línea transfiere el primer elemento del datalog a la matriz *Mat*, para a continuación asignar el número de elementos que contiene el datalog a la variable *iSup*.

```
Mat = PBrickCtrl.UploadDatalog(0, 1)
iSup = Mat(2, 0)
```

Para representar correctamente los datos hay que definir los límites del gráfico.

```
'define los límites del gráfico
iLimInfX = 0: iLimSupX = iSup       ‘x entre 0 y el número de elementos
iLimInfY = 500: iLimSupY = 850      ‘y con valores entre 500 y 850
iX = 0                              'x comenzará en la coordenada 0
```

El eje **x** contiene el número de elementos del datalog y el eje **y** las lecturas del sensor de luz para cada elemento. Una vez limpiado el cuadro de la imagen se define la escala: la primera coordenada la esquina superior izquierda, y la segunda la inferior derecha. La instrucción *Forecolor* no hace mas que establecer el color del gráfico, que en este caso es el rojo.

```
picGrafico.Cls
picGrafico.Scale (iLimInfX, iLimSupY)-(iLimSupX, iLimInfY)
picGrafico.ForeColor = QBColor(4)
```

La transferencia de datos ha de hacerse en lotes de 50 Esto quiere decir que si hay más de 50 elementos habrá que calcular cuantas transferencias extras (*iLotes*) hay que realizar. Para ello utilizaremos la siguiente expresión:

```
iLotes = Int(iSup / 50)       ‘parte entera del cociente
```

Si han de ser transferidos 69 datos, el datalog ha de ser transferido en dos lotes; un lote de 50 elementos y un segundo de 19 elementos. La variable *iLotes* debería ser igual a 1 para indicar que una transferencia extra es necesaria. A continuación se inicia un bucle For ... Next.

```
    For iCuenta = 0 To iLotes
        iInf = iCuenta * 50
        If iSup <= 50 Then
            Mat = PBrickCtrl.UploadDatalog(iInf, iSup)
        Else
            Mat = PBrickCtrl.UploadDatalog(iInf, 50)
        End If
        iSup = iSup – 50
            ’el código que falta aquí se describe más adelante
    Next iCuenta
```

El bucle comienza con el contador (*iCuenta*) a cero y se detiene cuando el contador alcanza el valor *iLotes*. *iInf* contiene la posición del elemento inicial que ha de ser transferido. Si han de ser tres los paquetes de elementos a transferir, inicialmente habrá de ser igual a cero, a continuación 50, y para acabar 100. La estructura de control If ... Then ... Else establece que, si han de ser transferidos 50 o menos elementos, se transfiera el número exacto, pero si los elementos a transferir son más de 50, se transferirá un paquete de 50 elementos, y se establecerá el número de datos restante a transferir (iSup = el último iSup menos 50), los cuales serán transferidos en el siguiente bucle del For.

Veamos el fragmento de código que falta:

```
        If IsArray(Mat) Then
            For i = LBound(Mat, 2) To UBound(Mat, 2)
                iX = iX + 1
                picGrafico.Line -(iX, Mat(2, i))
            Next i
        Else
            MsgBox "No es una matriz válida"
        End If
    Next iCuenta
```

Si por medio de la función *IsArray*() se comprueba que la matriz Mat es una matriz válida (es decir, transferida con éxito) entonces los límites inferior y superior de la matriz existen y se dibujará el gráfico por medio de un bucle For ... Next. Por cada valor del contador *i* dibujará un punto. *iX* contiene la coordenada **x** del último punto dibujado en el gráfico, la cual es incrementada en una unidad para dibujar el siguiente. La instrucción

```
picGrafico.Line -(iX, Mat(2, i))
```

sólo tiene una coordenada. Cuando sólo se proporciona una coordenada, esta define el punto final de la línea, y el punto inicial es aquél en el que la anterior línea ha finalizado (coordenada Xactual,Yactual).

9.6. Ejercicios

Si quieres modificar la cantidad de lecturas que se toman, modifica el número de veces en el bucle de la tarea 3. También se puede modificar la frecuencia con la que las lecturas son almacenadas en el datalog.

10 Comunicaciones entre robots

Si se dispone de más de un RCX, se pueden escribir programas que les permita comunicarse entre sí utilizando el puerto de infrarrojos. De este modo, se puede hacer que varios robots colaboren (o que peleen entre ellos). También se puede construir un gran robot utilizando dos RCX de tal manera que pueda tener seis motores y seis sensores

10.1. Contenidos de este capítulo

1. Métodos a utilizar en comunicaciones entre robots.
2. Robot master y esclavo. Definición de protocolos de comunicaciones.

10.2. Método de trabajo

Las comunicaciones entre dos RCX se hacen por medio de luz infrarroja utilizando el método *SendPBMessage*. Este método puede ser utilizado para transmitir un número comprendido entre 0 y 255 utilizando el transmisor de infrarrojos del RCX. Cualquier otro RCX que se encuentre cerca del RCX emisor, podrá recibir el mensaje y almacenarlo. El RCX que realiza la mayoría de las transmisiones se le llama generalmente *Master*, mientras que al receptor *Esclavo*. Para leer el mensaje recibido en el RCX utilizaremos el método Poll. El RCX puede también borrar el mensaje almacenado en su memoria interna utilizando la instrucción *ClearPBMessage*. Esta instrucción establece como valor interno del mensaje el "0".

10.3. Proyecto

El proyecto a realizar, es un sencillo programa que mostrará cómo un RCX, Master, envía un mensaje al otro (Esclavo).

- Empieza del modo habitual. Los datos del formulario los tienes en la tabla 10.1.
- Guarda tu proyecto como **RCXCom**.

Tipo de control	Propiedad	Valor
Form	Nombre Caption	FrmRCX_a_RCX Comunicaciones entre RCX
CommandButton	Nombre Caption	cmdMaster Transferir &Master
CommandButton	Nombre Caption	cmdEsclavo Transferir &Esclavo
CommandButton	Nombre Caption	cmdSondea &Sondea
CommandButton	Nombre Caption	cmdSalir S&alir
TextBox	Nombre Text	TxtSondeo (Dejarlo vacío)

Tabla 10.1

- Escribe el siguiente código:

```
Private Sub cmdSalir_Click()
      PBrickCtrl.CloseComm
      End
End Sub

Private Sub cmdSondea_Click()
      TxtSondeo.Text = Str(PBrickCtrl.Poll(PBMESS, 0))
End Sub

Private Sub cmdMaster_Click()
      With PBrickCtrl
            .SelectPrgm SLOT_3
            .BeginOfTask PRINCIPAL
                  .SendPBMessage CTE, 123
            .EndOfTask
      End With
End Sub

Private Sub cmdEsclavo_Click()
      With PBrickCtrl
            .SelectPrgm SLOT_4
            .BeginOfTask PRINCIPAL
                  .ClearPBMessage
                        'A la espera de un mensaje
                  .While PBMESS,0,IG,CTE,0
                        .Wait CTE,MS_50
                  .EndWhile
                  .PlaySystemSound SWEEP_DOWN_SOUND
            .EndOfTask
      End With
End Sub

Private Sub Form_Load()
      PBrickCtrl.InitComm
End Sub
```

- Guarda tu programa.
- Ejecuta tu programa
- Enciende uno de los RCX (llamaremos a este primero Master) y pulsa el botón *Transferir Master*.
- Apaga el *Master* y enciende el otro RCX (lo llamaremos *Esclavo*).
- Pulsa el botón *Transferir Esclavo*.
- Enciende otra vez el *Master*.
- Pulsa el botón **Run** del *Esclavo* y a continuación el del *Master*.

Si todo va bien, el *Esclavo* debe emitir un SYSTEM_SWEEP_DOWN.

- Apaga el *Master* y pulsa el botón *Sondea* del formulario. El número 123 debe aparecer en el cuadro de texto, lo que confirmará que el mensaje ha sido correctamente transmitido.

10.3.1. Descripción del programa

Cuando el programa *Master* se ejecuta, transmite el mensaje 123 y finaliza. Para entonces, el programa *Esclavo* estaba ejecutándose y esperando un mensaje. En el momento de recibirlo, emite un sonido y finaliza su ejecución.

Es muy importante definir un protocolo de comunicaciones, es decir, definir de modo previo el significado de cada mensaje.

10.3.2. Ejercicio

Con el objeto de que no surjan problemas, cuando el *Esclavo* recibe el mensaje, haz que confirme la recepción del mensaje contestando al *Master*, que estará esperando dicha confirmación (por ejemplo, el mensaje 1). Haz los cambios necesarios para que tenga este comportamiento.

10.4. Control remoto

Ahora controlaremos el robot del capítulo anterior por medio de otro RCX. El Master controlará el comportamiento del *Esclavo*. El *Esclavo* estará obligado a cumplir tres ordenes:

- Avanza (mensaje nº 1)
- Retrocede (mensaje nº 2)
- Detente (mensaje nº 3)

- Monta la base de robot (ver el apéndice A).
- Modifica el código de la siguiente manera:

```
Private Sub cmdMaster_Click()
      With PBrickCtrl
            .SelectPrgm SLOT_3

            .BeginOfTask PRINCIPAL
                  .ClearPBMessage
                  .SendPBMessage CTE, 1   ' adelante
                  .Wait CTE,SEG_3
                  .SendPBMessage CTE, 2   ' atrás
                  .Wait CTE,SEG_3
                  .SendPBMessage CTE, 3   ' parada
            .EndOfTask

      End With
End Sub

Private Sub cmdEsclavo_Click()
      With PBrickCtrl
            .SelectPrgm SLOT_4
            .BeginOfTask PRINCIPAL
                  .ClearPBMessage
                  .Loop CTE,SIEMPRE
                        ‘ en espera de mensajes
                        .While PBMESS, 0, IG, CTE, 0
                              .Wait CTE,MS_10
                        .EndWhile
                        ' Enciende los motores: adelante
                        .If PBMESS,0,IG,CTE,1
                              .SetFwd MOTOR_A + MOTOR_C
                              .On MOTOR_A + MOTOR_C
                        .EndIf
```

```
                        ' coloca aquí el código para que retroceda

                        ' coloca aquí el código para que se detenga
                        .ClearPBMessage
                    .EndLoop
                .EndOfTask
        End With
End Sub
```

- Guarda tu proyecto.
- Ejecuta tu proyecto.
- Repite el procedimiento de transferencia anterior.
- Ejecuta el programa en el *Esclavo*.
- Ejecuta el programa en el *Master*.

El robot *Esclavo* avanzará durante tres segundos y retrocederá durante otros tres antes de detenerse.

10.5. Ejercicios

1. Haz las modificaciones necesarias en el *Esclavo* para que confirme que recibe cada uno de los mensajes, pero esta vez, si el *Maestro* no recibe la confirmación en un determinado plazo de tiempo, el programa *Maestro* enviará de nuevo el mensaje al *Esclavo*. Tendrás que decidir un protocolo, es decir, qué número (0...255) será el mensaje de confirmación.
2. Coloca en una habitación varios robots que se muevan de modo aleatorio. Coloca un objeto en el suelo, y cuando un robot lo encuentre, que lo notifique al resto (en lugar de un objeto también puede ponerse un círculo negro sobre un fondo blanco). Cuando un robot lo encuentre lo notificará al resto y todos se detendrán.

11 Herramientas complementarias

En este capítulo vamos a ver algunas herramientas y técnicas de programación no vistas en capítulos anteriores.

11.1. Objetos Mutex

Cuando en un programa hay varias tareas, estas son ejecutadas en paralelo. Esto parece ideal, y efectivamente lo es, pero el concepto no es tan sencillo como parece. Por ejemplo, puede suceder que una tarea ordene que un motor gire en sentido de avance durante un determinado periodo de tiempo, y mientras el motor esté en marcha otra tarea ordene que el motor gire en sentido de retroceso. Esta situación puede ser deseable en algunos casos, pero no en otros. Esto puede evitarse utilizando un **mutex**.

Un objeto mutex es un objeto de sincronización que puede tener dos estados: ocupado (1) cuando está poseído por alguna tarea, o libre cuando no está poseído por ninguna tarea. Sólo una tarea puede poseer en cada momento un mutex, cuyo nombre proviene del hecho que su utilidad en coordinación de acceso **mut**uamente **ex**cluyente (co-ordinating mutually exclusive access) en recursos compartidos (por ejemplo, un motor).

Por ejemplo, para evitar que dos tareas controlen un motor a la vez, cada tarea espera a que el mutex esté libre antes de ejecutar el código que afecta al motor. Una vez que una tarea finaliza con el motor, el mutex es liberado

Los mutex se implementan por medio de variables. La variable tomará el valor 0 cuando el motor no esté utilizado por ninguna tarea. Cuando una tarea necesite utilizar el motor esperará a que el valor de la variable sea igual a cero. Cuando la variable es igual a cero, la tarea modifica su valor a 1, y sólo a partir de ese momento toma el control del motor. Cuando acaba de utilizar el motor, la tarea devuelve a la variable el valor inicial 0.

```
Private Sub cmdMutexEG_Click()

      Const MUTEX = 6                     ' La variable 6 será el mutex

      With PBrickCtrl

            .SelectPrgm SLOT_4
            .BeginOfTask PRINCIPAL
                  .SetVar MUTEX, CTE, 0   ' Inicialmente libre
                  .StartTask TAREA_1
                  .StartTask TAREA_2
            .EndOfTask

            .BeginOfTask TAREA_1
                              'si la tarea 1 quiere utilizar el motor...
                  .While ALD, MUTEX, IG, CTE, 1
                        .Wait CTE, MS_10
                  .EndWhile
                              ' toma como propio el mutex
                  .SetVar MUTEX, CTE, 1
                        ' aquí trabaja con el motor, y a continuación
```

```
                  ‘ libera el mutex
            .SetVar MUTEX, CTE, 0
      .EndOfTask

      .BeginOfTask TAREA_2
                  ‘si la tarea 2 quiere utilizar el motor...
            .While ALD, MUTEX, IG, CTE, 1
                  .Wait CTE, MS_10
            .EndWhile
                  ‘ toma como propio el mutex
            .SetVar MUTEX, CTE, 1
                  ‘ aquí trabaja con el motor, y a continuación
                  ‘ libera el mutex
            .SetVar MUTEX, CTE, 0
      .EndOfTask

   End With
End Sub
```

11.2. Subrutinas

Las subrutinas se utilizan para contener fragmentos de código que se utilizan frecuentemente. Por ejemplo, si a menudo arrancas y detienes un motor, puedes crear una subrutina. De este modo, siempre que necesites arrancar y detener un motor, será suficiente con hacer una llamada a la subrutina. Veámoslo por medio de un ejemplo

```
Private Sub cmdSubEG_Click()
   Const ONOFF = 3 ‘nombre de la subrutina
   With PBrickCtrl
      .SelectPrgm SLOT_4
      .BeginOfTask PRINCIPAL
            ' aquí código
         .GoSub ONOFF
            ' aquí más código
         .GoSub ONOFF
            ' aquí más código
      .EndOfTask

      .BeginOfSub ONOFF
         .On MOTOR_A
         .Wait CTE, SEG_3
         .Off MOTOR_A
      .EndOfSub
   End With
End Sub
```

No conviene hacer llamadas a una misma subrutina desde diferentes tareas, ya que ello puede provocar comportamientos inesperados.

Las subrutinas son realmente útiles en programas largos con largas tareas. Se pueden crear hasta 8 subrutinas en cada slot de programa. Sus nombres serán números comprendidos entre 0 y 7 (aquí también pueden utilizarse constantes para facilitar la lectura del programa).

En el programa del capítulo anterior se puede escribir una subrutina para esperar la llegada de un mensaje.

```
Private Sub cmdEsclavo_Click()
   Const ESPERA_MENS = 6         ' subrutina 6
```

```
	With PBrickCtrl

		.SelectPrgm SLOT_4
		' Chequea PSMESS en intervalos de 10 ms para comprobar si ha
		' llegado algún mensaje
		.BeginOfSub ESPERA_MENS
			.While PBMESS, 0, IG, CTE, 0
				.Wait CTE, MS_10
			.EndWhile
		.EndOfSub

		.BeginOfTask PRINCIPAL
			.ClearPBMessage
			.SetFwd MOTOR_A + MOTOR_C
			.Loop CTE, SIEMPRE
				'Espera el mensaje
				.GoSub MEZUAREN_ZAIN
				' aquí más código
				.ClearPBMessage
			.EndLoop
		.EndOfTask

	End With

End Sub
```

11.3. Temporizadores

El RCX tiene cuatro temporizadores independientes con una resolución de 100 ms. Estos pueden ser reiniciados individualmente utilizando el método *ClearTime*. Tan pronto como son reiniciados comienzan a contar otra vez a partir de cero. El temporizador puede tener un valor comprendido entre 0 y 32767, lo que quiere decir que el contador puede contar hasta 3276 segundos, lo que se corresponde aproximadamente con 55 minutos. Para reiniciar el temporizador utilizaremos la siguiente instrucción:

```
PBrickCtrl.ClearTimer TEMPOR_1
```

De este modo el temporizador empezará otra vez desde 0, añadiendo una unidad cada décima de segundo. Para ver el valor actual de un temporizador utilizaremos la orden Poll:

```
PBrickCtrl.Poll(TEMPOR, TEMPOR_1)
```

Recuerda que los temporizadores tienen una resolución de 100 ms y que la del método *Wait* es de 10 ms, es decir, el temporizador 10 tics por segundo mientras que *Wait* 100 tics por segundo. No utilizes las constantes definidas en el módulo RCXdatos.bas (por ejemplo MS_100) para comparar cualquier valor con valores contenidos en los temporizadores. La razón es que dichas constantes está definidas a partir de una resolución de 10 ms.

12 Bibliografía

"Lego Mindstorms programming with Visual Basic" David Hanley y Sean Hearne. http://emhain.wit.ie/~p98ac25/ (alguno de los apéndices originales no ha sido incluido en este manual)

"Controlling LEGO® Programmable Bricks Technical Reference" LEGO. http://mindstorms.lego.com/

"The Unofficial Guide to LEGO MINDSTORMS Robots". Jonathan B. Knudsen. O'REILLY 1999.

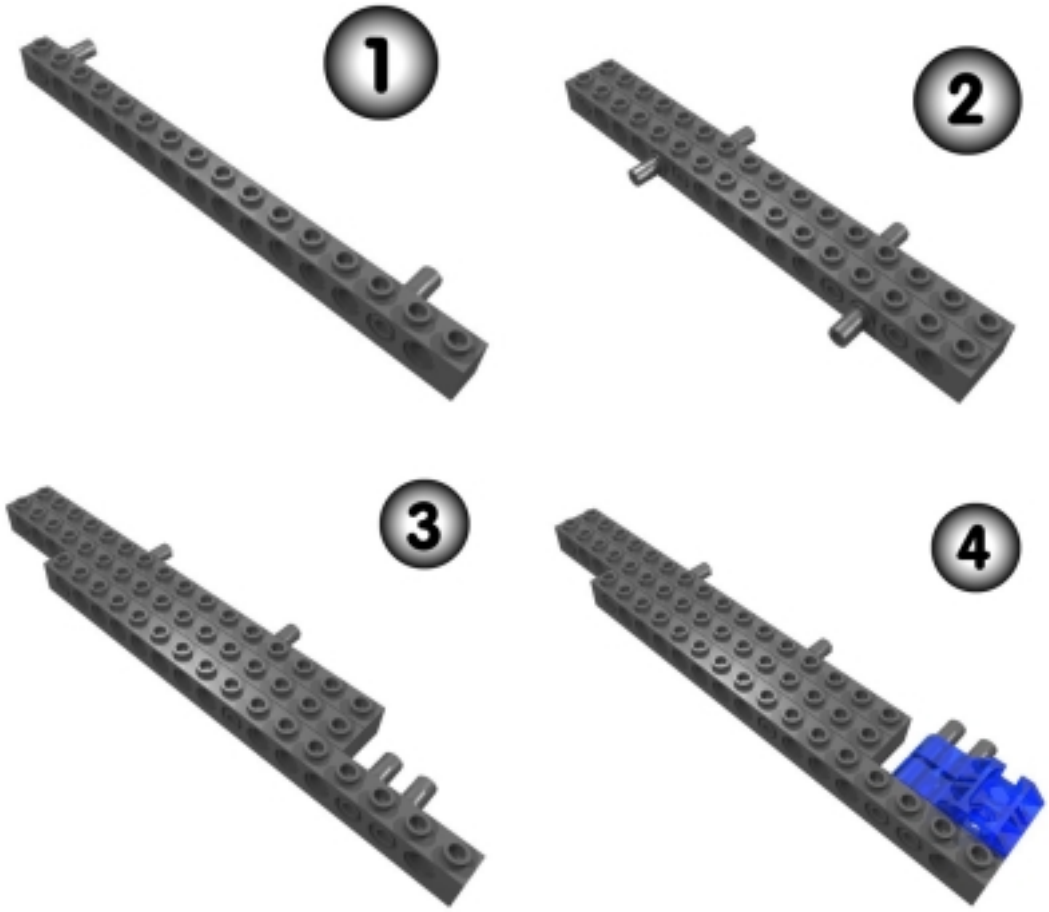
"Introducing Robotics with Lego Mindstorms" Robert Penfold. Bernard Babani LTD 2000

"More Advanced Robots with Lego MindStorms" Robert Penfold. Bernard Babani LTD 2000

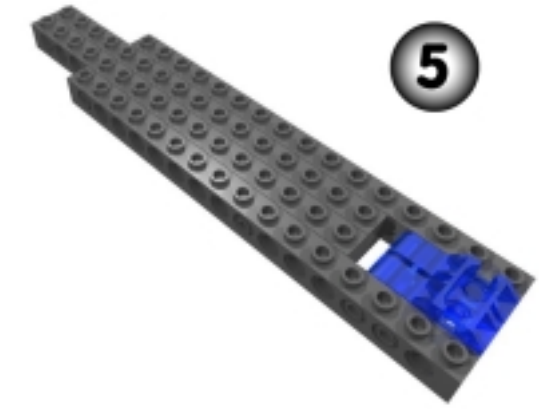
Apéndice A: Instrucciones de montaje

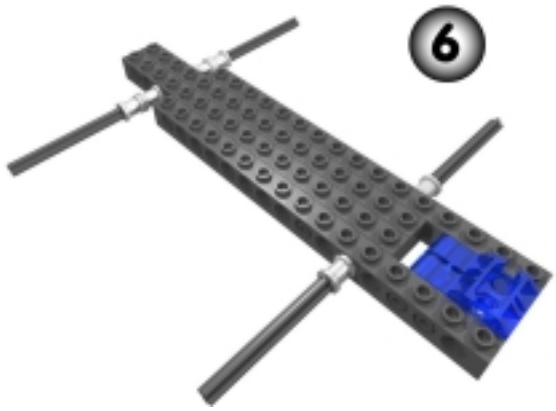
BASE DE ROBOT[9]

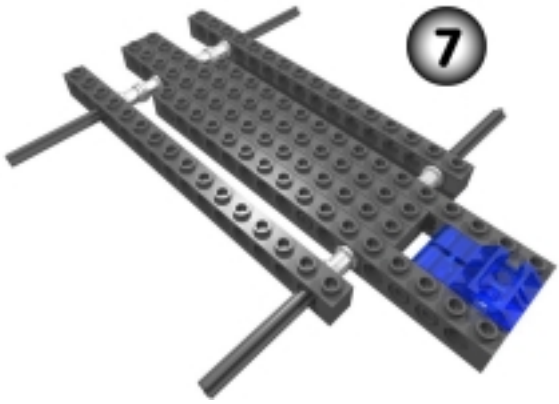
La base de robot utilizada en este manual es un modelo basado en la Constructopedia de LEGO MindStorms. Cualquier otro modelo semejante a este puede valer también, siempre y cuando el motor derecho se conecte a la salida C y el izquierdo a la A.

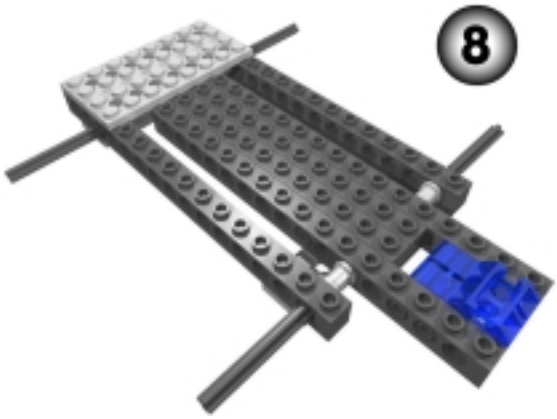


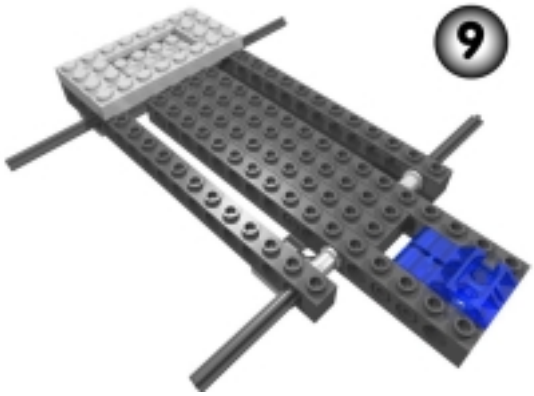
[9] Para hacer estas instrucciones he utilizado MLCAD. Los ficheros .DAT pueden encantarse junto a estas instrucciones.

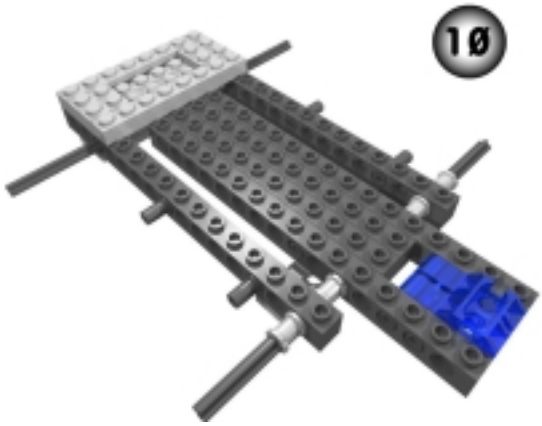

5

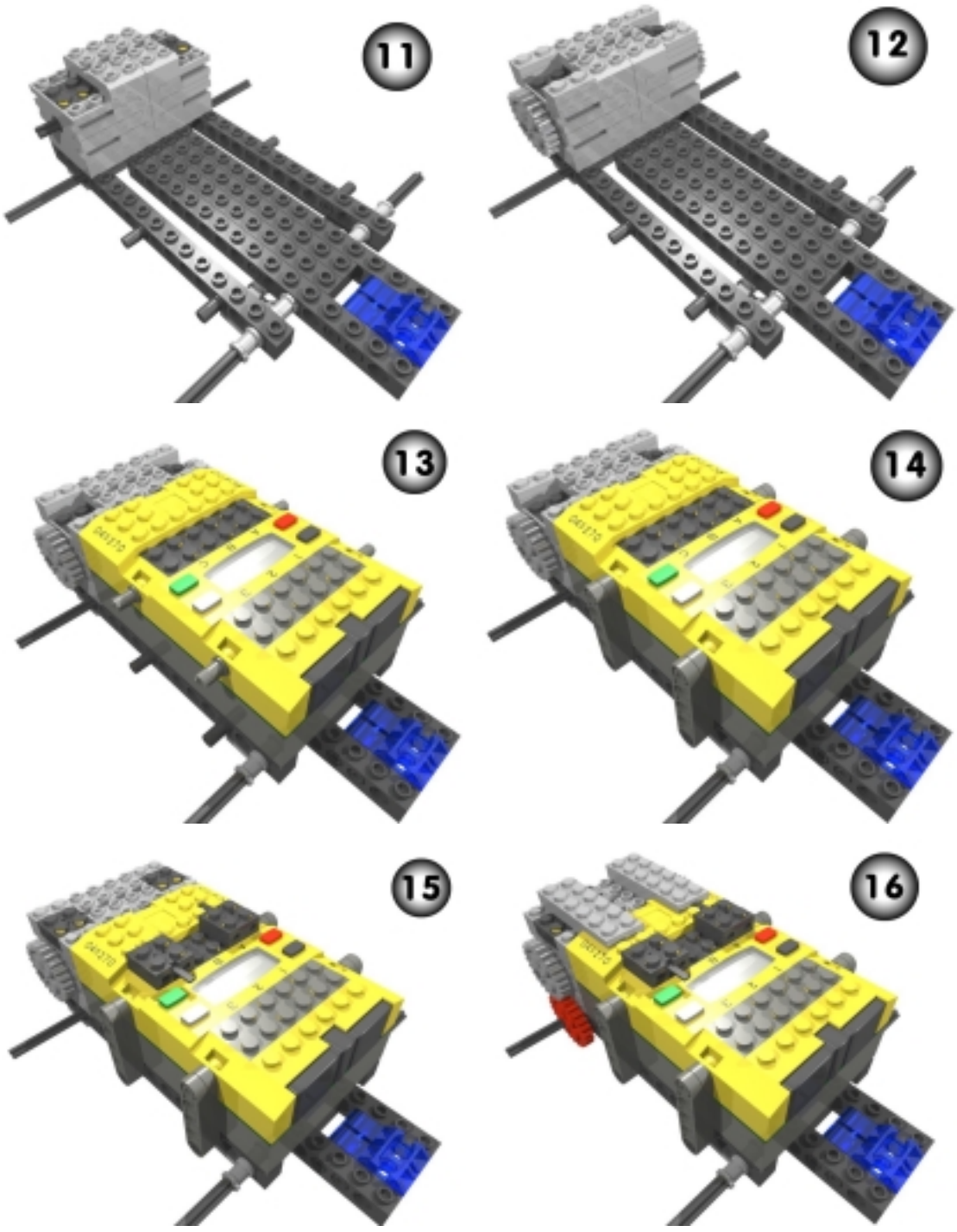

6


7


8

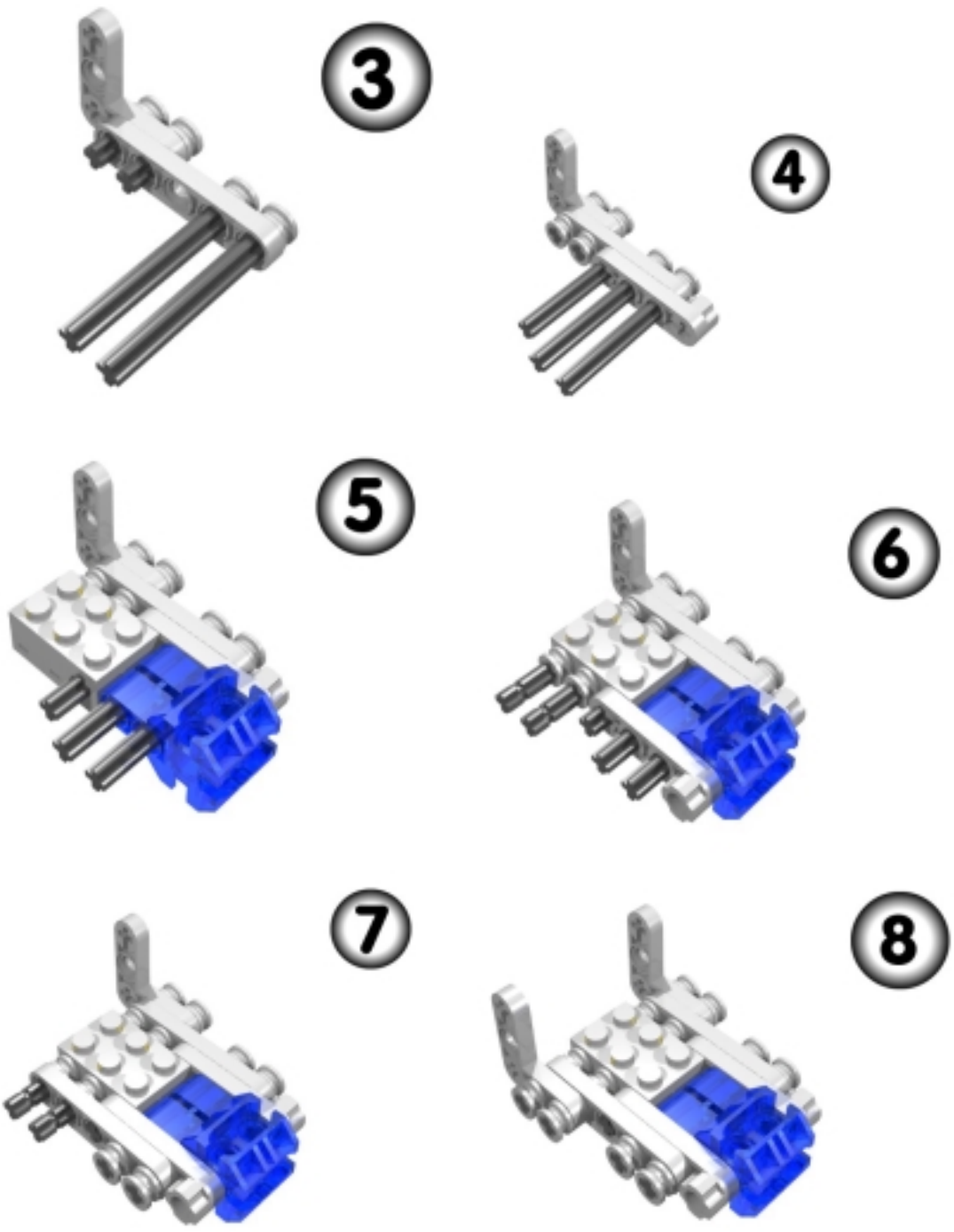

9

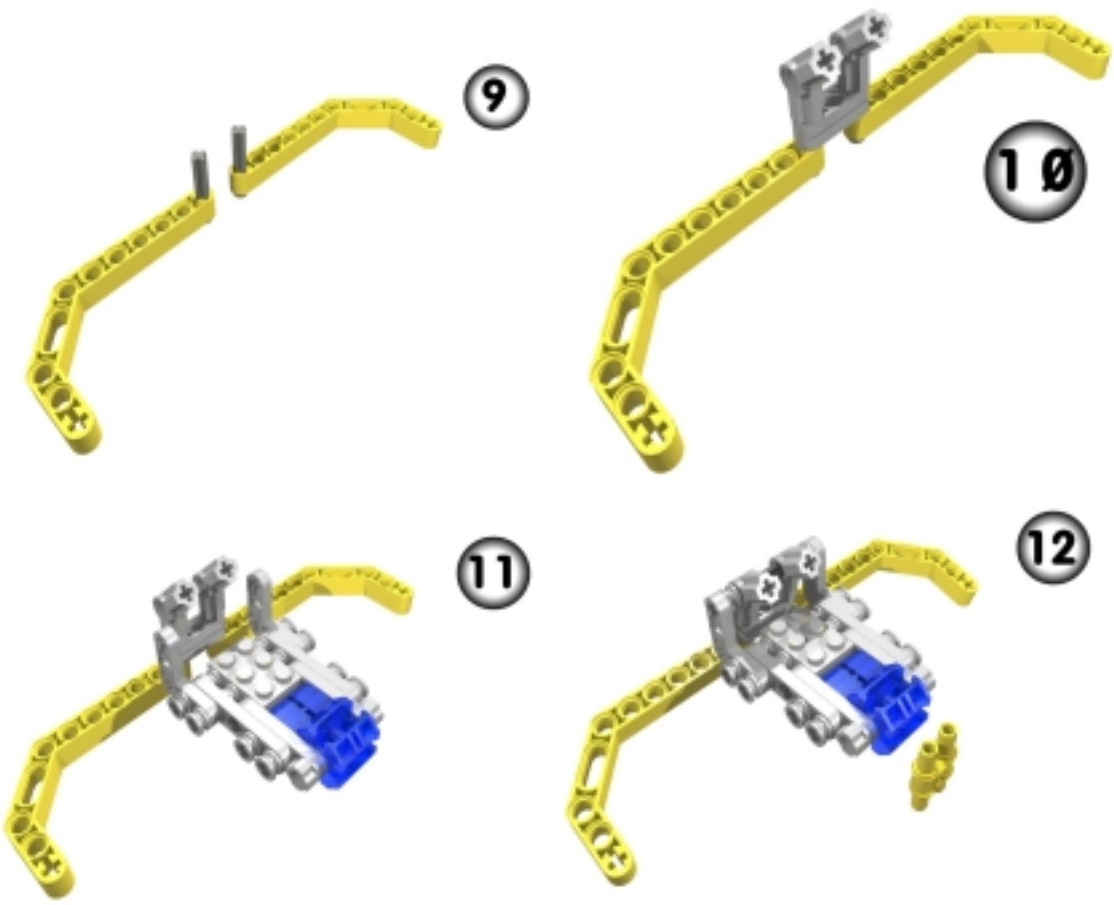

10

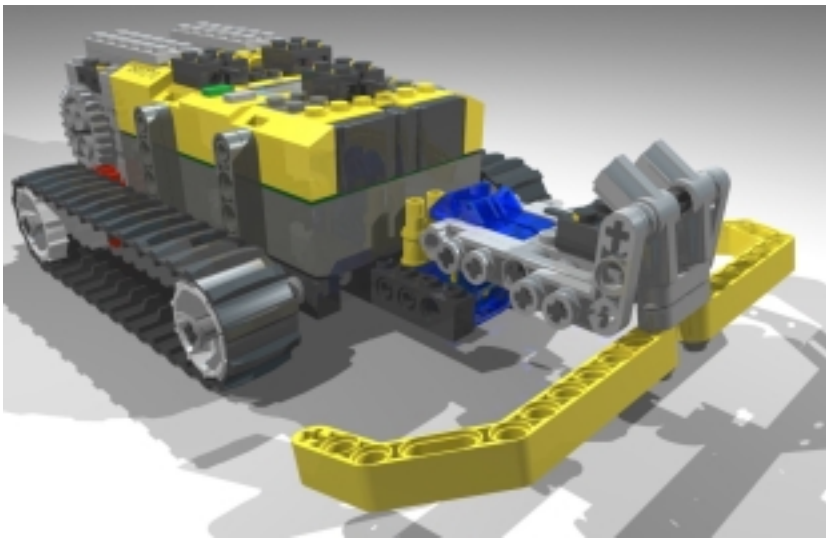

11
12
13
14
15
16

MÓDULO CON SENSOR DE CONTACTO

En los proyectos que requieren detectar obstáculos puedes añadir el siguiente módulo a la base de robot.

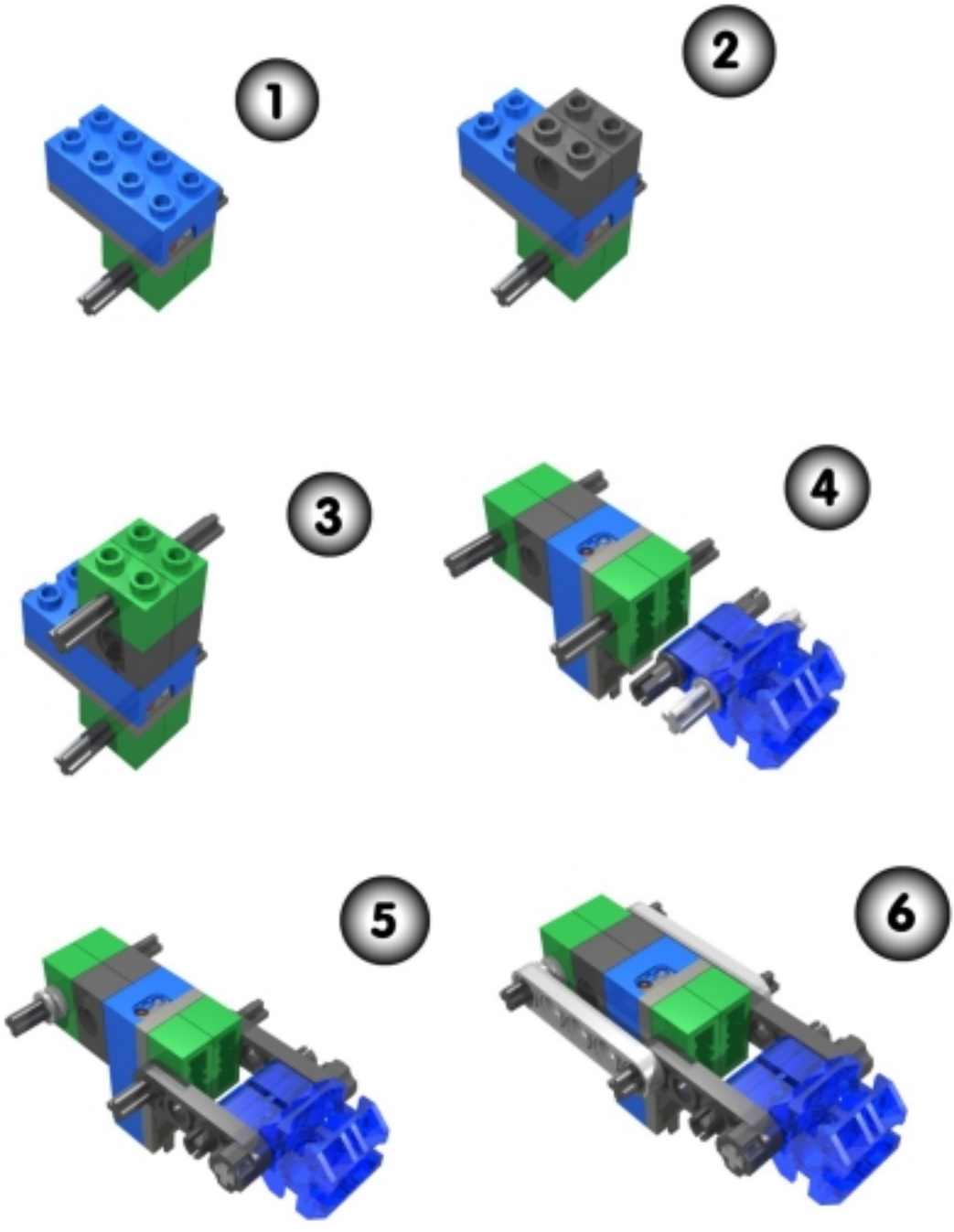

3
4
5
6
7
8

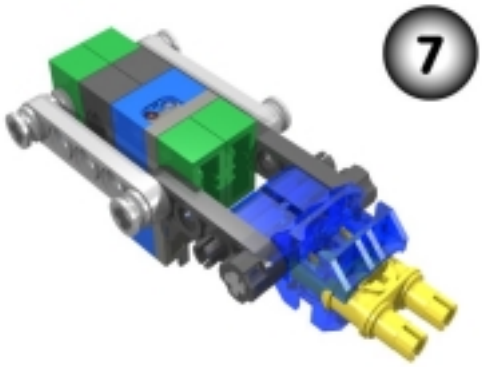

9
10
11
12



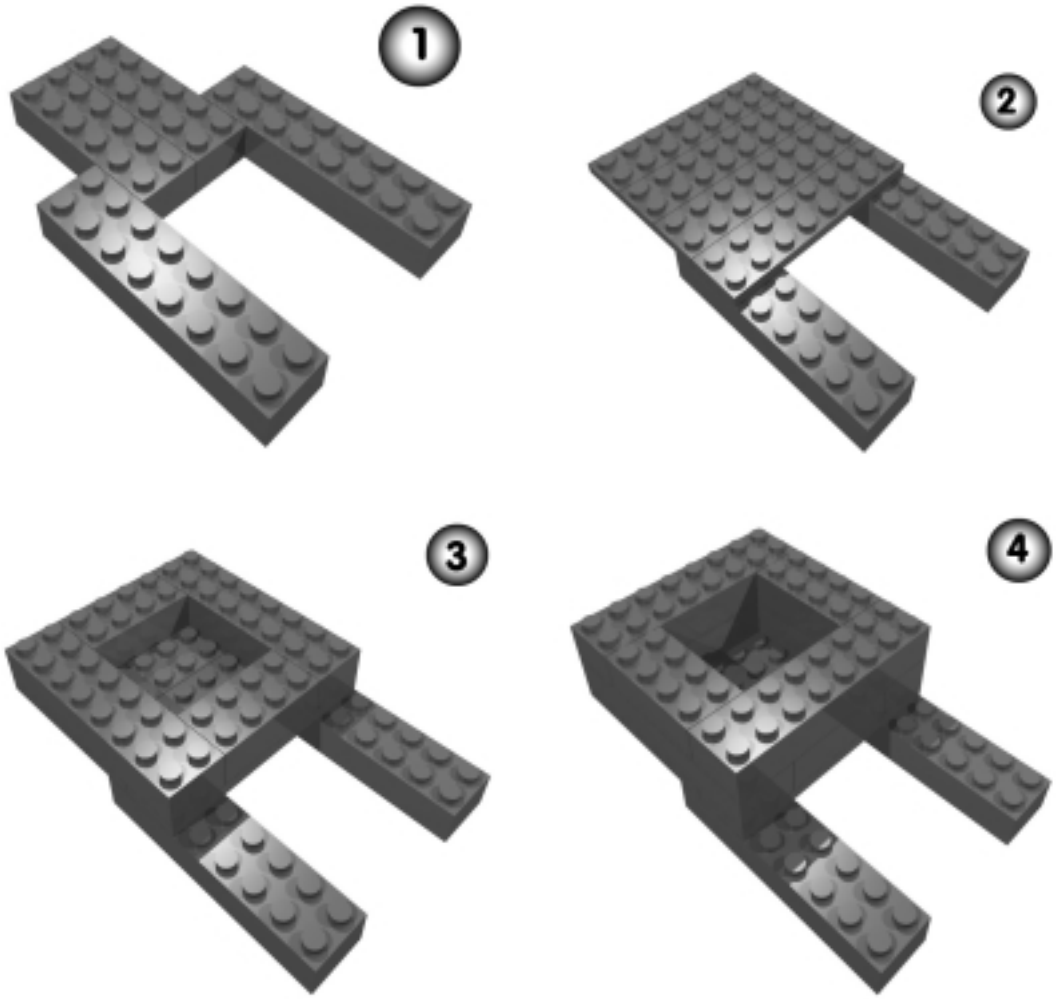
MÓDULO CON SENSOR DE LUZ

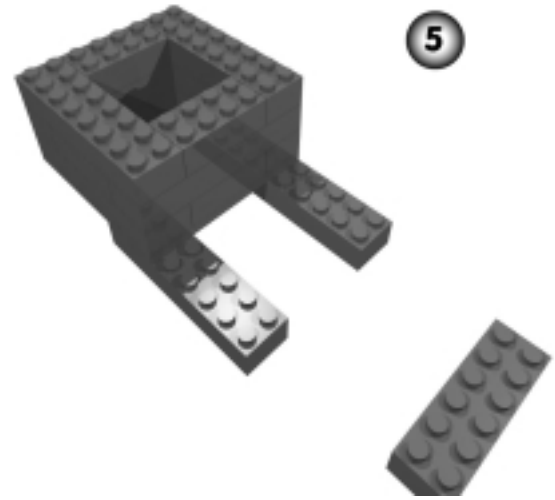
Este módulo es de diseño propio. A pesar de ser alargado no se desmonta con facilidad. Puedes utilizar este o cualquier otro.





DETECTOR DE COLOR




5


6

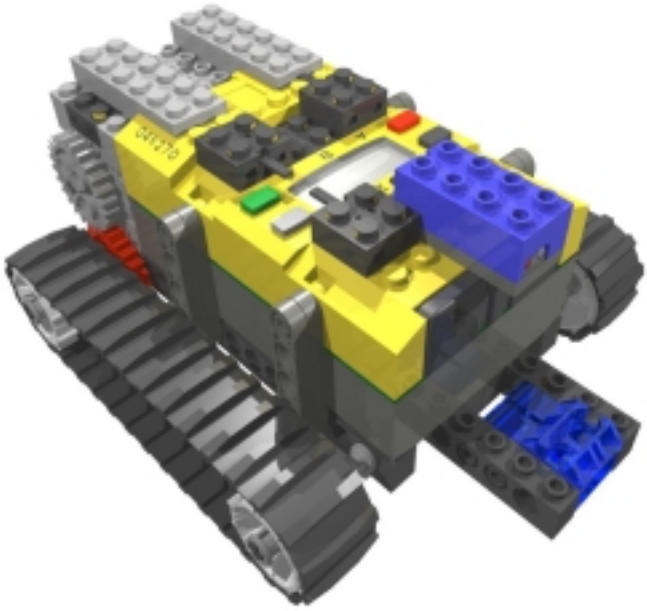

7


8


10

ROBOT “PROXIMIDAD”

El punto de partida para montar este robot es la base de robot anterior. Una vez montada la base añádele lo que le falta.



Anexo B: Módulo RCXdatos.bas

```
'====================================================================
' Programación de robots Lego MindStorms. Visual Basic.
' Modulo global
' 1.0 versión
'--------------------------------------------------------------------
' Declaración de nombres globales de constantes para el RCX
'====================================================================
Option Explicit
'====================================================================
' Introduce aquí o al inicio del programa tus propias constantes
'====================================================================

'====================================================================
' Slots de programas 0 - 4
'====================================================================
Public Const SLOT_1 = 0
Public Const SLOT_2 = 1
Public Const SLOT_3 = 2
Public Const SLOT_4 = 3
Public Const SLOT_5 = 4
'====================================================================
' Nombres de tareas - Modificar los nombres de tareas 1 - 9 por nombres
' más significativos. Para ello no es necesario que hagas las
' sustituciones aquí, puedes hacerlo al principio del programa
'====================================================================
Public Const PRINCIPAL = 0
Public Const TAREA_1 = 1
Public Const TAREA_2 = 2
Public Const TAREA_3 = 3
Public Const TAREA_4 = 4
Public Const TAREA_5 = 5
Public Const TAREA_6 = 6
Public Const TAREA_7 = 7
Public Const TAREA_8 = 8
Public Const TAREA_9 = 9
'====================================================================
' Sonidos de sistema
'====================================================================
Public Const CLICK_SOUND = 0
Public Const BEEP_SOUND = 1
Public Const SWEEP_DOWN_SOUND = 2
Public Const SWEEP_UP_SOUND = 3
Public Const ERRORE_SOUND = 4
Public Const SWEEP_FAST_SOUND = 5



'====================================================================
' Nombres de origen de datos
'====================================================================
Public Const VAR = 0
```

```
Public Const TEMPOR = 1
Public Const CTE = 2
Public Const MOTSTA = 3
Public Const RAN = 4
Public Const TACC = 5
Public Const TACS = 6
Public Const MOTCUR = 7
Public Const KEYS = 8
Public Const SENVAL = 9
Public Const SENTIPO = 10
Public Const SENMODO = 11
Public Const SENRAW = 12
Public Const BOOL = 13
Public Const RELOJ = 14
Public Const PBMESS = 15
'=============================================================================
' Nombres de sensores
'=============================================================================
Public Const SENSOR_1 = 0
Public Const SENSOR_2 = 1
Public Const SENSOR_3 = 2
'=============================================================================
' Nombres de temporizadores
'=============================================================================
Public Const TEMPOR_1 = 0
Public Const TEMPOR_2 = 1
Public Const TEMPOR_3 = 2
Public Const TEMPOR_4 = 3
'=============================================================================
' Nombres de tacómetro (sólo CyberMaster)
'=============================================================================
Public Const TACHO_IZQ = 0
Public Const TACHO_DCHO = 1
'=============================================================================
' Nombres de alcance
'=============================================================================
Public Const ALCANCE_CORTO = 0
Public Const ALCANCE_LARGO = 1
'=============================================================================
' Tipos de sensor
'=============================================================================
Public Const NO_TIPO = 0
Public Const TIPO_SWITCH = 1
Public Const TIPO_TEMP = 2
Public Const TIPO_LUZ = 3
Public Const TIPO_ANGULO = 4
'=============================================================================
' Modos de sensor
'=============================================================================
Public Const RAW_MODO = 0
Public Const BOOL_MODO = 1
Public Const TRANS_CONT_MODO = 2
Public Const PERIOD_CONT_MODO = 3
Public Const PORCENT_MODO = 4
Public Const CELSIUS_MODO = 5
Public Const FAHRENHEIT_MODO = 6
Public Const ANGULO_MODO = 7
'=============================================================================
' Nombres de motor (cadenas)
'=============================================================================
Public Const MOTOR_A = "0"
```

```
Public Const MOTOR_B = "1"
Public Const MOTOR_C = "2"
'=============================================================================
' Nombres de salida
'=============================================================================
Public Const SALIDA_A = "0"
Public Const SALIDA_B = "1"
Public Const SALIDA_C = "2"
'=============================================================================
' Operadores lógicos de comparación
'=============================================================================
Public Const MAYQ = 0   'Mayor que
Public Const MENQ = 1   'Menor que
Public Const IG = 2     'Igual
Public Const DIF = 3    'No igual a
'=============================================================================
' Miscellaneous
'=============================================================================
Public Const SIEMPRE = 0
'=============================================================================
' Constantes de tiempo (MS = milisegundo)
'=============================================================================
Public Const MS_10 = 1                      ' 10 ms
Public Const MS_20 = (2 * MS_10)            ' 20 ms
Public Const MS_30 = (3 * MS_10)            ' 30 ms
Public Const MS_40 = (4 * MS_10)            ' 40 ms
Public Const MS_50 = (5 * MS_10)            ' 50 ms
Public Const MS_60 = (6 * MS_10)            ' 60 ms
Public Const MS_70 = (7 * MS_10)            ' 70 ms
Public Const MS_80 = (8 * MS_10)            ' 80 ms
Public Const MS_90 = (9 * MS_10)            ' 90 ms
Public Const MS_100 = (10 * MS_10)          '100 ms
Public Const MS_200 = (20 * MS_10)          '200 ms
Public Const MS_300 = (30 * MS_10)          '300 ms
Public Const MS_400 = (40 * MS_10)          '400 ms
Public Const MS_500 = (50 * MS_10)          '500 ms
Public Const MS_700 = (70 * MS_10)          '700 ms
Public Const SEG_1 = (100 * MS_10)          ' 1 s
Public Const SEG_2 = (2 * SEG_1)            ' 2 s
Public Const SEG_3 = (3 * SEG_1)            ' 3 s
Public Const SEG_5 = (5 * SEG_1)            ' 5 s
Public Const SEG_10 = (10 * SEG_1)          '10 s
Public Const SEG_15 = (15 * SEG_1)          '15 s
Public Const SEG_20 = (20 * SEG_1)          '20 s
Public Const SEG_30 = (30 * SEG_1)          '30 s
Public Const MIN_1 = (60 * SEG_1)           ' 1 mn
```

Anexo C: Referencia técnica

El documento “Controlling LEGO® Programmable Bricks Technical Reference” recoge información técnica sobre los métodos del control spirit.ocx. En dicho documento se encuentra la información necesaria para programar el CyberMaster y el RCX. En las líneas siguientes se ofrece información sobre algunos de los métodos, pero no es mas que un resumen. Así que si necesitas más información, ya sabes donde encontrarla.

On (MotorList)

Pone en marcha los motores de la lista.

MotorList: cadena de texto formada por los nombres de los motores que hay que poner en marcha. Sus valores pueden ser “1”, “2” y “3”, pero también pueden utilizarse las constantes que sustituyen dichos valores: MOTOR_A...

Métodos relacionados: El método Off(MotorList) detiene los motores en modo freno.

Ejemplo: La instrucción PBrickCtrl.On “02” pondrá en marcha los motores A y C.

Poll(Source, Number)

Por medio de esta instrucción se obtiene información del RCX, por ejemplo, lecturas de sensores, valor de las variables o el estado de los motores.

Source y Number: origen del dato y valor correspondiente (véase la tabla 5.3).

Ejemplo: La instrucción PBrickCtrl.Poll 0,7 se utiliza para conocer el valor de la variable número 7.

SetFwd (MotorList)

Este método establece el sentido de giro de los motores de la lista como sentido de avance. No tiene ningún efecto en el resto de propiedades.

MotorList: cadena de texto que define la lista de motores. Sus valores pueden ser “0”, “1” y “2”, pero también pueden utilizarse las constantes que los sustituyen: MOTOR_A...

Métodos relacionados: el método SetRew(MotorList) establece el sentido como retroceso y el método AlterDir(MotorList) cambia el sentido de giro por el opuesto.

Ejemplo: SetFwd “02”

SetPower (MotorList, Source, Number)

Establece el nivel de potencia de los motores.

MotorList: cadena de texto que define la lista de motores. Sus valores pueden ser “0”, “1” y “2”, pero también pueden utilizarse las constantes que los sustituyen: MOTOR_A...

Source eta Number: origen de datos y valor.

Ejemplo: SetPower (“012”, 0, 15) establece como nivel de potencia para los tres motores en el nivel que define el contenido de la variable 15.

SetSensorMode (Number, Mode)

Define el formato de las lecturas del sensor.

Number: número de entrada (o constante que lo sustituya).

Type: modo de sensor (tabla 5.4)

Ejemplo: PbrickCtrl.SetSensormode 0, 0 configura el sensor conectado en la entrada 0 (la que en el RCX tiene el número 1) en el modo Raw, por lo que las lecturas tendrán un valor comprendido entre 0 y 1023. Puede utilizarse la constante SENSOR_1 en lugar dell número 0.

SetSensorType (Number, Type)

Se utiliza para definir qué tipo de sensor se ha conectado.

Number: número de entrada

Type: tipo de sensor (tabla 5.2)

Ejemplo: PbrickCtrl.SetSensorType 0, 4 se utiliza cuando se conecta en la entrada 0 un sensor angular.

SetVar (VarNo As Integer, Source As Integer, Number As Integer)

Asigna un valor a una variable.

VarNo: número que identifica la variable (entre 0 y 31) o constante que lo sustituye.

Source: origen del valor, que puede ser desde una constante a la lectura de un sensor.

Number: valor a asignar a la variable, que puede ser desde un valor constante hasta la identificación de un sensor.

Ejemplo: SetVar 2, 0, 5 asigna a la variable 2 el contenido de la variable 5.

Apéndice D: Transferencia de programas al RCX con control de errores

Cuando se transfieren programas al RCX el evento *DownloadDone* informa sobre el éxito de la operación.

- Si el programa es transferido al RCX sin errores, *ErrorCode* es igual a 1.
- Si ocurre algún error, *ErrorCode* es igual a 0.

El siguiente código puede ser incorporado a cualquier proyecto. Si se desea incorporarlo a todo programa transferible al RCX, conviene recogerlo en el código de la plantilla con la que se trabaje.

El formulario de dicha plantilla debería tener el siguiente aspecto:

Figura D.1: formulario

```
' Todas las variables han de ser declaradas
Option Explicit
Dim blnEspera As Boolean
Dim blnTransferOK As Boolean

Private Sub cmdTransferir_Click()
      blnEspera = True
      ' Escribir el código a transferir al RCX aquí
End Sub

Private Sub cmdSalir_Click()
      PBrickCtrl.CloseComm
      End
End Sub

Private Sub Form_Load()
      PBrickCtrl.InitComm
      blnEspera = False
      blnTransferOK = False
End Sub
```

```
Private Sub PBrickCtrl_AsyncronBrickError(ByVal Number As Integer,_
Description As String)
      If (blnEspera) Then
            While (blnEspera)
                  DoEvents
            Wend
            MsgBox "Asynchronous Brick Error: " + Str(Number) + " " +_
            Description, vbCritical, "Transferencia fallida"
      Else
            MsgBox "Asynchronous Brick Error: " + Str(Number) + " " +_
            Description, vbCritical, "Transferencia fallida"
      End If
End Sub

Private Sub PBrickCtrl_DownloadDone(ByVal ErrorCode As Integer, ByVal
DownloadNo As Integer)
      If ErrorCode = 0 Then
            blnTransferOK = True
            'MsgBox "Transferencia hecha correctamente"
      Else
            'MsgBox "Transferencia fallida"
      End If
      blnEspera = False
End Sub

Private Sub PBrickCtrl_DownloadStatus(ByVal timeInMS As Long, ByVal _
sizeInBytes As Long, ByVal taskNo As Integer)
      If (blnEspera) Then
            While (blnEspera)
                  DoEvents
            Wend
            If (blnTransferOK) Then
                  OutputStats timeInMS, sizeInBytes, taskNo
                  blnTransferOK = False
            End If
      Else
            If (blnTransferOK) Then
                  OutputStats timeInMS, sizeInBytes, taskNo
                  blnTransferOK = False
            End If
      End If
End Sub

' Información sobre la transferencia en un cuadro de texto
Public Sub OutputStats(Time As String, Size As String, Task As String)
      Dim LFCR As String
      LFCR = Chr(13) + Chr(10)
      MsgBox "Tiempo: " + Time + LFCR + "Tamaño: " + Size + LFCR +"Tarea_
      Número: " + Task, vbInformation, "Transferencia con éxito"
End Sub
```

Descripción del código

El propósito de este código es el siguiente:

- Detectar los errores que puedan suceder durante la transferencia del programa.
- No hacer nada mientras el evento *DownloadDone* esté ejecutándose.

- Mostrar las estadísticas del programa sólo si el programa ha sido correctamente transferido.

Si el control ActiveX envía cualquier evento y obliga a que cualquier ventana de diálogo sea abierta, el resto de controles ActiveX de la aplicación de Visual Basic desaparecerán.

Digamos que por ejemplo una instrucción de cuadro de mensaje aparece en el procedimiento de evento *DownloadDone* event, y que también aparece otro por medio del procedimiento de evento *AsyncronBrickError*. Si ocurriese un error durante la transferencia del programa, el cuadro de mensaje colocado en el monitor por el procedimiento *DownloadDone* debería desaparecer, y el cuadro de mensaje de *AsyncronBrickError* se abriría.

El anterior código no permite hacer nada a los procedimientos *AsyncronBrickError* o *DownloadStatus* mientras el código del procedimiento *DownloadDone* esté ejecutándose (cuando blnEspera = True), y el procedimiento *DownloadStatus* sólo ofrecerá estadísticas en el monitor si la transferencia se ha realizado con éxito (blnTransferOK = True).

Apéndice E: Sondeo de la configuración de los motores

Sondear (Poll) el motor para obtener información sobre él es diferente al resto de las opciones (por ejemplo a sondear un sensor). La razón de esto es que la información está empaquetada. Esto significa que para identificar la información en el número entero que el motor devuelve, es necesario convertirlo en un número binario (8 bits en este caso).

En la tabla 5.3 del capítulo cinco se ofrecen los significados de cada bit.

Tipo de control	Propiedad	Valor
Form	Nombre Caption	frmMotorPoll Sondeo de los motores
CommandButton	Nombre Caption	cmdSondear &Sondear
CommandButton	Nombre Caption	cmdSalir S&alir
Text Box	Nombre Text	txtDec (dejarlo vacío)
Text Box	Nombre Text	txtBin (dejarlo vacío)
Text Box	Nombre Text	txtOnOff (dejarlo vacío)
Text Box	Nombre Text	txtFreno (dejarlo vacío)
Text Box	Nombre Text	txtSalida (dejarlo vacío)
Text Box	Nombre Text	txtDireccion (dejarlo vacío)
Text Box	Nombre Text	TxtPotencia (dejarlo vacío)

Tabla E.1

Una vez diseñado el formulario a partir de los datos de la tabla E.1 escribe el código que se presenta a continuación. El siguiente código muestra cómo utilizar el número entero devuelto por medio del método *Poll* .

```
' Todas las variables han de ser declaradas
Option Explicit

Private Sub cmdSalir_Click()
      PBrickCtrl.CloseComm
```

```
      End
End Sub

Private Sub cmdSondear_Click()
      Dim strStatus As String
      Dim iMotor As Integer                     'contiene el valor entero
      Dim bMotor As String                      'contiene el valor binario

      iMotor = PBrickCtrl.Poll(MOTSTA, 0)
      txtDec = Str(iMotor)

      bMotor = Bin(iMotor)                      'Valor binario
      txtBin = bMotor

'Busca el nivel inferior
      strStatus = Mid(bMotor, 6, 3)             'obtiene los bits 0-2
      txtPower = Str(BintoDec(strStatus))       'valor decimal

'Buscar dirección
      If Val(Mid(bMotor, 5, 1)) = 1 Then        'obtiene el bit 3
            txtDireccion = "Avance"             'if = 1 => Fwd(Avance)
      Else
            txtDireccion = "Retroceso"          'if = 0 => Rev(Retroceso)
      End If

' Buscar número de salida
      strStatus = Mid(bMotor, 3, 2)             'obtiene los bits 4-5
      txtOutput = Str(BintoDec(strStatus))      'valor decimal

      ' Freno / Detenido[10]
      If Val(Mid(bMotor, 2, 1)) = 1 Then        'obtiene el bit 6
            txtBrake = "Frenado"                'if = 1 => Brake
      Else
            txtBrake = "Detenido"               'if = 0 => Float
      End If
      'ON / OFF
      If Val(Mid(bMotor, 1, 1)) = 1 Then        'obtener el bit 7
            txtOnOff = "ON"                     'if = 1 => On
      Else
            txtOnOff = "OFF"                    'if = 0 => Off
      End If
End Sub

Private Sub Form_Load()
      PBrickCtrl.InitComm
End Sub

Public Function Bin(Number As Integer) As String
      Dim strBit As String
      Dim iPos As Integer
      Dim iNumero As Integer
      iNumero = Number
      For iPos = 7 To 0 Step -1
            If >= (2 ^ iPos) Then
                  strBit = strBit + "1"
                  iNumero = iNumero - (2 ^ iPos)
            Else
```

[10] Por detenido se entiende que se ha utilizado para pararlo la orden Float, es decir, suspendiendo la alimentación y dejando que se detenga solo.

```
                strBit = strBit + "0"
            End If
        Next
        Bin = strBit                                    'devuelve el resultado
End Function

Public Function BintoDec(Number As String)
        Dim iLongitud As Integer
        Dim bNumero As String
        Dim iHam As Integer
        Dim iPos As Integer

        iHam = 0

        bNumero = Number
        iLongitud = Len(bNumero)

        For iPos = iLongitud To 1 Step -1
            If Mid(bNumero, 1, 1) = "1" Then
                iHam = iHam + (2 ^ (iLongitud - 1))
            End If
            bNumero = Mid(bNumero, 2, iLongitud)
            iLongitud = iLongitud - 1
        Next

        BintoDec = iHam

End Function
```

Descripción del código

Cada vez que el botón *Sondear* es pulsado, un entero es devuelto conteniendo información sobre los motores, pero dicha información está empaquetada. Es por ello necesario convertir el valor entero en una cadena binaria (string).

```
bMotor = Bin(iMotor) 'Valor binario
```

Por ejemplo si el entero pasado por medio de la función Bin es 79, se convertirá en la cadena "01001111". Esta es la representación binaria del número decimal 79.

Veamos ahora cómo interpretar dicha información. La siguiente tabla describe el significado de cada bit:

7	6	5 4	3	2 1 0
On /Off	Frenado / Float	Número de salida	Sentido de giro CW[11]/CCW	Nivel de potencia

Tabla E.2

[11] CW: movimiento a favor de las agujas del reloj. CCW: movimiento en contra del sentido de las agujas del reloj.

En nuestro caso la interpretación será la siguiente:

0	1	00	1	111
Parado (OFF)	frenado	Salida 0	Sentido de las agujas del reloj (CW)	Potencia=7

Tabla E.3

Veamos bit a bit cómo se interpreta esta cadena binaria.

Potencia del motor:

```
strStatus = Mid(bMotor, 6, 3) ' Obtiene el valor de los bits 0, 1 y 2
txtPotencia = Str(BintoDec(strStatus))    ' valor decimal
```

La función Mid devuelve a partir de la posición dada por el segundo argumento tantos caracteres como señala el tercer argumento. Por ejemplo, la instrucción Mid("Lego Mindstorms", 6, 4) devolvería la cadena “Mind”.

La instrucción strStatus = Mid(bMotor, 6, 3) devuélvelos tres caracteres de la cadena binaria que se encuentran a partir de la posición sexta. Para un número binario, esto sería los bits 2, 1 y 0. Este valor comunica el nivel de potencia del motor seleccionado en el formulario binario. Para obtener el valor decimal correspondiente se utiliza la función *BinToDec*.

```
txtPotencia = Str(BintoDec(strStatus))          ' valor decimal
```

Esta función obtiene un número decimal a partil de otro binario. El cuadro de texto txtPotencia mostrará este valor entero.

Ejemplo

Si el valor devuelto por el método Poll es 79, y queremos extraer los últimos tres bits para conocer el nivel de potencia del motor, habremos de recorrer la siguiente secuencia:

79 → “01001111” → “111” → 7

Completo → Binario → Bits requeridos → Completo

Sentido de giro del motor:

Para determinar la dirección de giro del motor necesitamos extraer el carácter quinto (bit tres) de la cadena.

```
If Val(Mid(bMotor, 5, 1)) = 1 Then  ‘ extraer el tercer bit
      txtDireccion = "Avance" ‘sentido a favor de las agujas del reloj
Else
      txtDireccion = "Retroceso"    ‘sentido en contra de las agujas del_
                                    reloj
End If
```

Apéndice F: otras opciones para utilizar VB

Hay otras opciones para utilizar Visual Basic como son “Visual Basic for Aplications” y “BrickCommand”.

Visual Basic for Aplications (VBA)

VBA puede encontrarse en las aplicaciones de Microsoft Office, AutoCAD 2000 y en las últimas versiones de Corel Draw.

Acompaña a este manual el documento Word “VBA plantilla” que facilita comenzar a trabajar directamente con VBA. En el mismo documento de ofrecen algunas instrucciones.

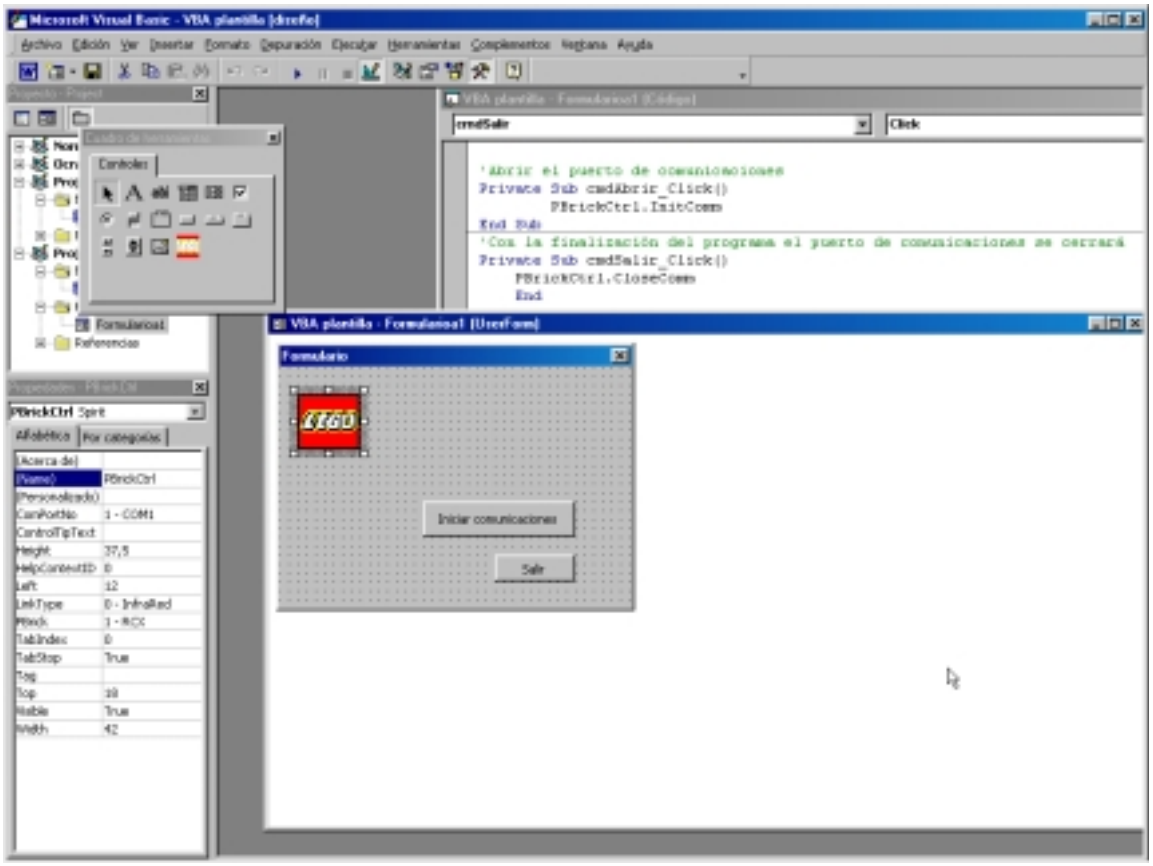


Figura F.1: VBA

Tiene ciertas limitaciones si lo comparamos con Visual Basic, pero es muy útil. Además, si se tiene alguna de las aplicaciones anteriormente mencionadas no supone ningún gasto adicional.

BrickCommand

Esta aplicación puede encontrarse en Internet. Por medio de ella pueden editarse programas en Visual Basic y transferirlos a continuación al RCX.

Se encuentra en http://www.geocities.com/Area51/Nebula/8488/ y la última versión es la 2.0. es gratuita. Los programas que pueden editarse con esta aplicación han de ser transferibles al RCX, ya que no acepta control directo desde el ordenador (sólo puede hacerse por medio del panel que ofrece el programa). Dado que está diseñado para programas transferibles al RCX, podemos olvidar los métodos *InitComm* y *CloseComm*.

En la edición hay que tener dos cosas en cuenta:

- Los argumentos de los métodos han de escribirse entre paréntesis sin dejar ningún espacio.
- A los métodos sin argumento también hay que añadirles paréntesis.

Figura F.2:
BrickCommand

Apéndice H: Transferencia del firmware al RCX

Para poder comunicarse con el RCX desde el PC es necesario que previamente el firmware haya sido transferido. Si al encender el RCX no aparece el reloj (inicialmente muestra el valor 00.00) y no puede utilizarse el botón View será necesario transferir el firmware. La transferencia puede hacerse por medio del software suministrado con LEGO MindStorms o LEGO Dacta, o por medio del programa en Visual Basic que se ofrece a continuación.

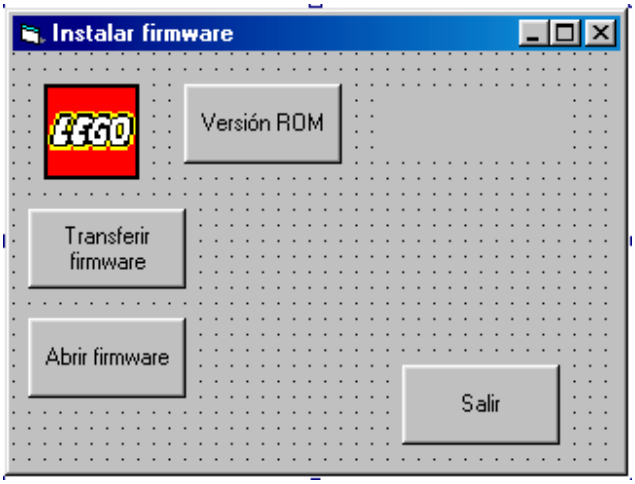


Figura H.1 formulario

Si tras ejecutar el siguiente procedimiento los cinco últimos caracteres de la cadena de texto recuperada son 00.00, el RCX no tiene firmware.

```
 ‘ Obtener la versión ROM
Private Sub cmdRomVersión_Click()
      lblRom.Caption = PBrickCtrl.UnlockPBrick
End Sub
```

Para instalar el firmware en l RCX lo primero que hay que hacer es transferirlo:

```
 ‘ Transferir el firmwarea
Private Sub cmdDownloadFirmware_Click()
      PBrickCtrl.DownloadFirmware "C:\Archivos de Programa\LEGO_
      MINDSTORMS\Firm\firm0309.lgo"[12]
End Sub

‘ Éxito de la transferencia
Private Sub PBrickCtrl_DownloadDone(ByVal ErrorCode As Integer, ByVal_
DownloadNo As Integer)
      If ErrorCode = 0 Then
            MsgBox "El Firmware ha sido correctamente transferido",_
            vbInformation
```

[12] Esta es la versión que corresponde al RCX 1.5. La versión de firmware correspondiente al RCX 2.0 es la firm0328.lgo da, pero da problemas para ser transferida por medio de este método.

```
      Else
            MsgBox "Firmware mal transferido", vbCritical
      End If
End Sub

Private Sub Form_Load()
      PBrickCtrl.InitComm
End Sub

Private Sub cmdSalir_Click()
      PBrickCtrl.CloseComm
      End
End Sub
```

La transferencia durará varios minutos y cuando finalice un mensaje aparecerá en el monitor. Ahora hay que abrir el firmware. Para ello podemos utilizar el siguiente procedimiento.

```
 'Abrir el Firmwarea
Private Sub cmdAbrirFirmware_Click()
      lblFirmware.Caption = PBrickCtrl.UnlockFirmware("Do you byte, when_
 I knock?")             ' Nota: este texto no puede modificarse
End Sub
```

La etiqueta lblFirmware mostrará el siguiente texto:

"This is a LEGO Control OCX communicating with a LEGO PBrick!"

Si falla el texto será el siguiente:

"Unlock failed"

Ahora el RCX ya está preparado para guardar programas.

Índice