Programando Robots Lego usando NQC

(Version 3.03, Oct 2, 1999)

por Mark Overmars

Department of Computer Science
Utrecht University
P.O. Box 80.089, 3508 TB Utrecht
the Netherlands

Prefacio.

El Lego MindStorms y CyberMaster son nuevos y maravillosos juguetes con los culales se puede construir y programar una amplia variedad de robots que cumplan complicadas tareas. Desafortunadamente, el software que viene con el juguete es visualmente atractivo pero algo limitado en lo que respecta a su funcionalidad. Por lo que con el solo se pueden programar una cantidad limitada de tareas. Para liberar el verdadero poder de los robots , Se necesita un ambiente de programacion diferente . NQC es un lenguaje de programacion , escrito por Dave Baum, Que fue especialmente diseñado para robots Lego. Si tu nunca has escrito un programa antes,no te preocupes. NQC es en verdad facil de usar y esta guia te dira todo acerca de como utilizar NQC. De hecho, programar los robots en NQC es mucho mas facil que programar una computadora normal asi que esta es la oportunidad de convertirse en un programador de una manera facil.

Para escribir programas aun más fácil existe el RCX Command Center (Centro de Comandos RCX). Esta utilidad te ayuda escribir tus programas, enviarlos al robot, y encender o detener el robot. El RCX Command Center trabaja casi como un procesador del texto en ingles, pero con algunos extras. Esta guía didáctica usará el RCX Command Center (versión 3.0 o más alto) como ambiente de programacion. Puedes descargarlo GRATIS de la red en la siguiente dirección:

http://www.cs.uu.nl/people/markov/lego/

El RCX Command Center corre en PC´s con Windows ('95, '98, NT). (Asegurate que ejecutaste el software que viene con el juego de Lego por lo menos una vez, antes de usar el RCX Command Center. El software de Lego instala ciertos componentes que el RCX Command Center necesita.) El lenguaje NQC también puede usarse en otras plataformas. Puedes descargarlo de la red en la siguiente direccion:

http://www.enteract.com/~dbaum/lego/nqc/

La mayor parte de esta guía didáctica también funciona para las otras plataformas (asumiéndo que estas usando NQC versión 2.0 o más alto), sólo que en algunas versiones faltan algunas de las herramientas o los colores.

En esta guía didáctica yo asumo que tienes el RIS (Robotic Inventnion System) de Lego MindStorms. La mayoría de los contenidos también funciona para los robots de CyberMaster aunque algunas funciones no están disponibles para esos robots. También los nombres de los motores son diferentes por lo que tendrás que cambiar los ejemplos un poco para hacerles trabajar.

Reconocimientos

Me gustaría agradecer Dave Baum por su investigacion de NQC. También muchas gracias a Kevin Saddi por escribir una primera versión de la primera parte de esta guía didáctica.

Contenido

Prefacio — ***1***
Reconocimientos — 2

Contenido — ***3***

I. Escribiendo tu primer programa — ***5***
Construyendo un robot — 5
Empezando con el RCX Command Center — 5
Escribiendo el programa — 6
Ejecutando el programa — 7
Errores en tu programa — 7
Cambiando la velocidad — 8
Resumen — 8

II. Un programa más interesante — ***9***
Haciendo Giros — 9
Repitiendo Ordenes — 9
Agregando un comentario — 10
Resumen — 11

III. Usando Variables — ***12***
Moviendose en espiral — 12
Números al azar — 13
Resumen — 13

IV. Estructuras de control — ***14***
La declaración if — 14
La declaración do — 15
Resumen — 15

V. Sensores — ***16***
Esperando por un sensor — 16
Actuando en un sensor del toque — 16
Sensores de luz — 17
Resumen — 18

VI. Tareas y subrutinas — ***19***
Tareas — 19
Subrutinas — 20
Funciones en linea (Inline) — 20
Definiendo macros — 21
Resumen — 22

VII. Haciendo Musica — ***23***
Sonidos preprogramados — 23
Escuchando musica — 23
Resumen — 24

VIII. Más sobre los motores — ***25***
Deteniendo suavemente — 25
Comandos avanzados — 25
Cambiando la velocidad del motor — 26
Resumen — 26

IX. Más sobre los sensores — ***27***
Modo del sensor y tipo — 27
El sensor de rotación — 28
Colocando multiples sensores en una sola entrada — 28
Haciendo un sensor de proximidad — 29

Resumen 30

X. Tareas paralelas *31*

Un mal programa 31

Deteniendo y reiniciando tareas 31

Usando semaforos 32

Resumen 33

XI. Comunicación entre los robots *34*

Dando ordenes 34

Elegiendo a un líder 35

Precauciones 35

Resumen 36

XII. Más comandos *37*

Cronómetros 37

La pantalla 37

Datalogging 38

XIII. NQC referencia rápida *39*

Declaraciones 39

Condiciones 39

Expresiones 39

Funciones RCX 40

Constantes de RCX 41

Glosario 42

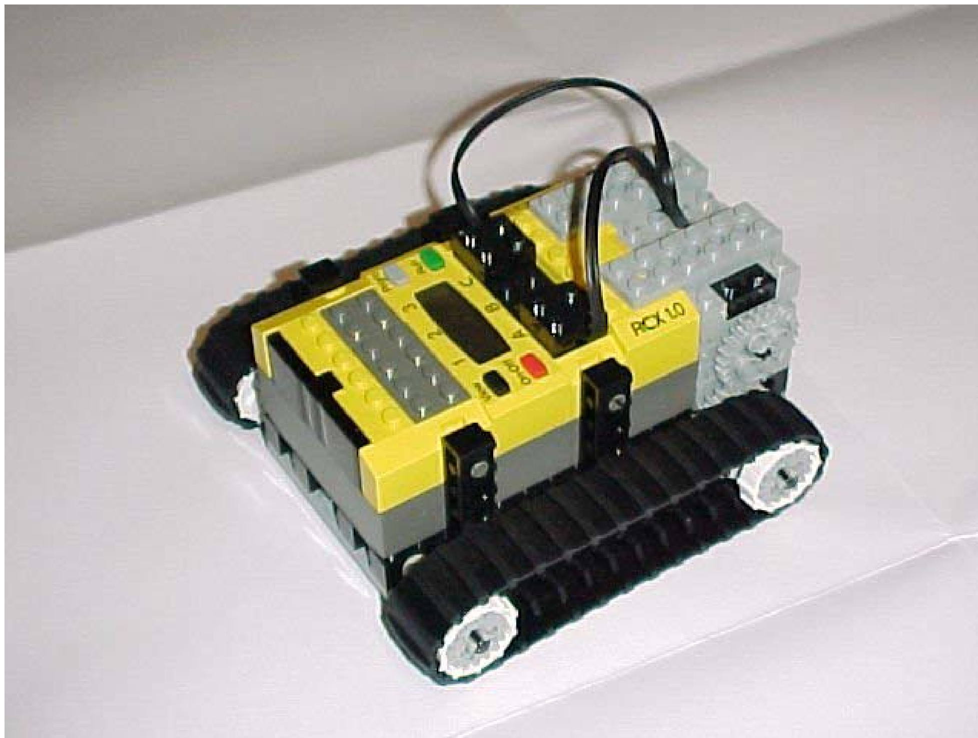
XIV. Comentarios finales *43*

I. Escribiendo tu primer programa

En este capítulo yo te mostraré cómo escribir un programa sumamente simple. Nosotros vamos a programar un robot que se mueva hacia adelante durante 4 segundos, luego se mueva hacia atras durante otros 4 segundos, y despues se detenga. No muy espectacular pero te presentará la idea básica de programación. Y te mostrará que fácil es esto. Pero antes de que nosotros podamos escribir un programa, primero necesitamos un robot.

Construyendo un robot

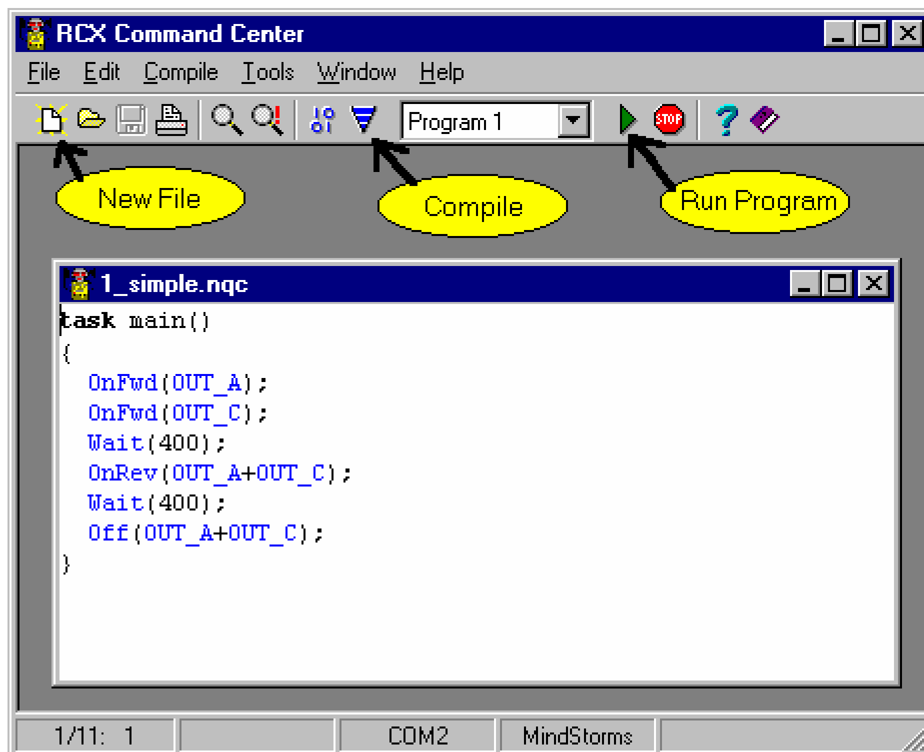
El robot que usaremos a lo largo de esta guía didáctica es una versión simple del robot super secreto que se describe en la página 39-46 de tu constructopedia. Nosotros usaremos sólo el chasis basico. Quita el frente entero con los dos brazos y los sensores del toque. También, conecta los motores ligeramente diferente de tal forma que los alambres se conectan al RCX quedan hacia afuera. Esto es importante para que tu robot se ponga en marcha en la dirección correcta. Tu robot debe parecerse a este:



También asegurate de que el puerto infra-rojo este conectado correctamente a tu computadora y que el switch en la parte baja este colocado para distancias largas. (Tal vez podrías verificar con el software de RIS que el robot está funcionando bien.)

Empezando con el RCX Command Center

Nosotros escribimos nuestros programas usando RCX Command Center. Empiézalo pulsando el botón dos veces en el icono RcxCC. (Yo asumo que el RCX Command Center ya lo instalaste. Si no, descargalo del sitio de internet (ve el prólogo), descomprimelo, y colocalo en cualquier directorio que te guste.) El programa te preguntará dónde localizar el robot. Enciende el robot y oprime OK. El programa (probablemente) encontrara el robot automaticamente. Ahora la interface del usuario aparece como se muestra debajo (sin una ventana).



La interface se parece a un editor de texto normal con los menús usuales como : los botones para abrir ,guardar , imprimir, editar archivos, etc.,. Pero hay también algunos menús especiales para compilar y transmitir programas al robot y para recibir información del robot. pero puedes ignorar éstos por el momento.

Nosotros vamos a escribir un nuevo programa. Así que aprieta el botón New File (archivo nuevo) para crear una nueva ventana vacía.

Escribiendo el programa

Ahora teclea en la ventana programa siguiente:

```
task main()
{
  OnFwd(OUT_A);
  OnFwd(OUT_C);
  Wait(400);
  OnRev(OUT_A+OUT_C);
  Wait(400);
  Off(OUT_A+OUT_C);
}
```

Esto puede verse un poquitin complicado al principio, así que hagamos un analisis. Los programas en NQC consisten de tareas. Nuestro programa tiene simplemente una tarea (task), nombrada `main`. Cada programa necesita tener una tarea llamada .Main (principal) qué es la que será ejecutada por el robot. Aprenderás más sobre las tareas en el Capítulo VI. Una tarea consiste en varios comandos, también llamados declaraciones. Hay corchetes alrededor de las declaraciones para que este claro que todoslos comandos contenidos dentro pertenecen a esta tarea. Cada declaración acaba con un punto y coma. De esta manera está claro donde una declaración acaba y donde la próxima declaración empieza. Asi que una tarea cualquiera , en general , se vera como sigue:

```
task main() //esta es la tarea principal
{
  statement1; // declaracion 1
  statement2; // declaracion 2
    …
}
```

Nuestro programa tiene seis declaraciones (statements). Analizemos cada una.

```
OnFwd(OUT_A);
```

Esta declaración dice al robot encender la salida A (output A) , es decir, el motor esta conectado a la salida etiquetada como A en el RCX, esto hara que el robot mueva hacia adelante con la velocidad máxima, a menos que tú primero le indiques la velocidad. Mas adelante veremos cómo hacer esto.

```
OnFwd(OUT_C);
```

Es la misma declaración pero ahora nosotros encendemos el motor conectado en C. Después de estas dos declaraciónes , ambos motores están corriendo, y el robot se mueve adelante.

```
Wait(400);
```

Ahora es tiempo para esperar durante algún tiempo. Esta declaración nos dice que esperemos por 4 segundos. El argumento, es decir, el número entre los paréntesis, da el número de "ticks.". Cada tick es 1/100 de un segundo. asi que puedes definir en el programa de una forma muy precisa cuánto tiempo debe esperar el robot. asi que durante 4 segundos, el programa no hace nada y el robot se continúa moviendo hacia adelante.

```
OnRev(OUT_A+OUT_C);
```

El robot ahora se ha movido suficientemente lejos para que nosotros le digamos que corra en dirección inversa, eso es, en reversa.. Nota que nosotros podemos encender ambos motores usando OUT_A+OUT_C en seguida como argumento. Nosotros también podríamos combinar las primeras dos declaraciones de esta manera.

```
Wait(400);
```

De nuevo nosotros esperamos por 4 segundos.

```
Off(OUT_A+OUT_C);
```

Y finalmente nosotros apagamos ambos motores.

Ése es el programa entero. Mueve ambos motores hacia adelante durante 4 segundos, entonces al revés durante 4 segundos, y finalmente los apaga.

Probablemente hayas notado los colores de las letras al teclear en el programa. Ellos aparecen automáticamente. estos colores significan que todo lo que esta en azul es una orden para el robot, o una indicación de un motor u otra cosa que el robot sabe. La palabra task (tarea) está en negrita porque es una palabra importante (reservada) en NQC. Otras palabras importantes aparecen en negrita así como veremos después. Los colores son útiles para ver que no hiciste ningún error mientras tecleabas.

Ejecutando el programa

Una vez que has escrito un programa, necesita ser compilado (es decir, cambiarlo a un codigo que el robot pueda entender y ejecutar) y envíado al robot que atravez de el puerto infrarojo de RIS (a esto se le llama "descargar" el programa). Hay un botón que hace las dos cosas al mismo tiempo (ve la figura arriba en donde dice Compile). Aprieta este botón y, asumiéndo que no hubo ningún error al teclear en el programa,este se compilará correctamente y se descargará. (Si hay errores en tu programa seras notificado; ve debajo.)

Ahora puedes ejecutar tu programa. Con este fin aprieta el botón verde(esto hara que el programa descargado se ejecute) en tu robot o, más fácilmente, aprieta el botón de "run program" en la ventana del Command Center (ve la figura arriba). ¿Hace el robot lo que esperaste? Si no, los alambres probablemente se encuentran mal conectados.

Errores en tu programa

Al teclear en programas hay una oportunidad razonable que haya algunos errores. El compilador nota los errores y los informa a ti al fondo de la ventana, como en la figura siguiente:

```
1_errors.nqc
task main()
{
  OnFwd(OUT_D);
  OnFwd(OUT_C);
  Wait(400);
  OnRev(OUT_A+OUT_C);
  Wait(400);
  Of(OUT_A+OUT_C);
}

line 3: Error: undefined variable 'OUT_D'
```

Selecciona el primer error automáticamente (nosotros hemos escrito mal el nombre del motor). Cuando hay más errores, puedes pulsar el botón en los mensajes del error para ir a ellos. Nota que amenudo os errores al inicio del programa son la causa de otros errores en otros lugares. asi que mejor corrige solamente los primeros errores y entonces compila el programa de nuevo. También nota que el codigo de color ayuda mucho para evitar errores. Por ejemplo, en la última línea nosotros tecleamos `Of` en lugar de `Off`. Debido a que éste es un comando desconocido no está coloreado de azul.

Hay también errores que no son encontrados por el compilador. Si nosotros hubiéramos tecleado `OUT_B` esto habría sido inadvertido porque ese motor existe (aunque nosotros no lo usamos en el robot). Por lo que si el robot exhibe una conducta inesperada, probablemente hay algo malo en tu programa.

Cambiando la velocidad

Como habras notado, el robot se movió bastante rápido. Esto es por que al programar esta predefinido que el robot se mueva tan rápido como pueda. Para cambiar la velocidad puedes usar el Comando `SetPower()` . El poder para el motor es un número entre 0 y 7. 7 es el mas rápido, 0 el más lento (pero el robot todavía se moverá). Aquí esta una nueva versión de nuestro programa en el cual los movimientos del robot son mas lentos:

```
task main()
{
  SetPower(OUT_A+OUT_C,2);
  OnFwd(OUT_A+OUT_C);
  Wait(400);
  OnRev(OUT_A+OUT_C);
  Wait(400);
  Off(OUT_A+OUT_C);
}
```

Resumen

En este capítulo escribiste tu primer programa en NQC, usando RCX Command Center. Ahora deberias saber teclear en un programa, cómo transmitirlo al robot y cómo permitir que el robot ejecute el programa. RCX Command Center puede hacer muchos cosas mas. Para averiguar sobre ellas, lee la documentación que viene con él. Esta guía didáctica se tratará principalmente del lenguaje NQC y sólo menciónara rasgos del RCX Command Center cuando realmente los necesites.

También aprendiste algunos aspectos importantes del idioma NQC. En primer lugar, aprendiste que cada programa tiene una tarea nombrada principal (`task main`) que es siempre es ejecutada por el robot. También aprendiste los cuatro órdenes del motor más importantes: `OnFwd ()`, `OnRev()`, `SetPower()` y `Off()`. Finalmente, aprendiste sobre la declaracion `Wait ()`.

II. Un programa más interesante

Nuestro primer programa no era muy espectacular. Así que permítenos intentar hacerlo más interesante. Nosotros haremos esto en varios pasos y presentaremos algunos rasgos importantes de nuestro idioma de programación NQC.

Haciendo Giros

Puedes hacer que tu robot de un giro deteniendo o invirtiendo la dirección de uno de los dos motores. Aquí esta un ejemplo. Tecléalo, descargalo a tu robot y permítele correr. El robot deberia avanzar un momento y despues d hacer un giro a la derecha de 90 grados.

```
task main()
{
  OnFwd(OUT_A+OUT_C);
  Wait(100);
  OnRev(OUT_C);
  Wait(85);
  Off(OUT_A+OUT_C);
}
```

tal vez tengas que probar algunos números ligeramente diferentes que 85 en la segunda declaracion `Wait ()` para hacer un giro preciso de 90 grados. Esto depende del tipo de superficie en la cual el robot corre. En lugar de cambiar esto en el programa es más fácil de usar un nombre para este número. En NQC puedes definir valores constantes como esta mostrado en el programa siguiente.

```
#define TIEMPO_DE_MOVERSE      100
#define TIEMPO_DE_DARVUELTA    85

task main()
{
  OnFwd(OUT_A+OUT_C);
  Wait(TIEMPO_DE_MOVERSE);
  OnRev(OUT_C);
  Wait(TIEMPO_DE_DARVUELTA);
  Off(OUT_A+OUT_C);
}
```

Las primeras dos líneas definen dos constantes. Éstas pueden usarse ahora a lo largo del programa. Definiendo constantes es bueno por dos razones: hace el programa más leíble, y es más fácil de cambiar los valores. Nota ese RCX Command Center da a las declaraciones definidas su propio color. Como nosotros veremos en Capítulo VI, también puedes definir otras instrucciones ademas de constantes.

Repitiendo ordenes

Permítenos ahora intentar escribir un programa que hace el robot maneje en un cuadrado. Para manejar en un cuadrado se necesita : manejar adelante, dar un giro 90 grados, manejar adelante de nuevo, dar un giro de 90 grados, etc.,. Nosotros podríamos escribir el pedazo anterior de código cuatro veces pero esto puede hacerse mucho más fácil con la declaracion **`repeat`** ().

```
#define TIEMPO_DE_MOVERSE   100
#define TIEMPO_DE_DARVUELTA 85

task main()
{
  repeat(4)
  {
    OnFwd(OUT_A+OUT_C);
    Wait(TIEMPO_DE_MOVERSE);
    OnRev(OUT_C);
    Wait(TIEMPO_DE_DARVUELTA);
  }
  Off(OUT_A+OUT_C);
}
```

El número delante de la declaración repeat, entre los paréntesis, indica qué tan a menudo algo debe repetirse. Las declaraciones que deben repetirse se ponen entre los corchetes , asi como las declaraciones en una tarea. Nota que, en el programa anterior, nosotros también separamos con sangria las declaraciones. Esto no es necesario, pero hace el programa más leíble.

Como un ejemplo final, hagamos que el robot maneje 10 veces en un cuadrado. Aquí esta el programa:

```
#define TIEMPO_DE_MOVERSE   100
#define TIEMPO_DE_DARVUELTA    85

task main()
{
  repeat(10)
  {
    repeat(4)
    {
      OnFwd(OUT_A+OUT_C);
      Wait(TIEMPO_DE_MOVERSE);
      OnRev(OUT_C);
      Wait(TIEMPO_DE_DARVUELTA);
    }
  }
  Off(OUT_A+OUT_C);
}
```

Hay ahora una declaracion repeat dentro de otra. Nosotros llamamos este una declaracion repeat "anidada". Puedes anidar declaraciones repeat tanto como te guste. Echa una mirada cuidadosa a los corchetes y la separacion con sangria usada en el programa. La tarea empieza en el primer corchete y termina en el último. La primera declaracion repeat inicia en el segundo corchete y termina en el quinto. La segunda declaracion repeat anidada inicia en el tercer corchete y termina en el cuarto. Como puedes ver los corchetes vienen en pares, y el pedazo entre los corchetes es lo que repetimos.

Agregando Comentarios

Para hacer tu programa aun mas leíble, es bueno agregar algún comentario a él. Siempre que escribas // en una línea, el resto de esa línea se ignora y puede usarse para los comentarios. Un comentario largo puede ponerse entre /* y */. Los comentarios están coloreados de verde en el RCX Command Center. El programa completo podría parecerse al que sigue:

```
/*  10 CUADROS

    por Mark Overmars

Este programa hace al robot correr en 10 cuadros
*/

#define TIEMPO_DE_MOVERSE    100       // Tiempo de movimiento
recto
#define TIEMPO_DE_DARVUELTA 85      // Tiempo para girar en 90
grados

task main()
{
  repeat(10)                // Hace los 10 cuadros
  {
    repeat(4)
    {
      OnFwd(OUT_A+OUT_C);
      Wait(TIEMPO_DE_MOVERSE);
      OnRev(OUT_C);
      Wait(TIEMPO_DE_DARVUELTA);
    }
  }
  Off(OUT_A+OUT_C);         // Ahora apaga los motores
}
```

Resumen

En este capítulo aprendiste el uso de la declaración repeat (repite) y el uso del comentario. También viste la función de corchetes anidados y el uso del espaciado. Con todo lo que sabes hasta ahora que puedes hacer que el robot siga toda clase de caminos. Es un buen ejercicio intentar y escribir algunas variaciones de los programas mostrados en este capítulo antes de continuar con el próximo capítulo.

III. Usando variables

Las variables forman un aspecto muy importante de cada lenguaje de programación. Las variables son lugares de memoria en las que nosotros podemos guardar un valor. Nosotros podemos usar ese valor en lugares diferentes y tambien podemos cambiarlo. Permíteme describir el uso de variables usando un ejemplo.

Moviendose en espiral

Asumiendo que nosotros queremos adaptar el programa anterior de tal forma que el robot maneje en una espiral. Esto puede ser logrado haciendo que el tiempo en el cual decimos al robot que espere sea más grande para cada próximo movimiento recto. Es decir, nosotros queremos aumentar el valor de `TIEMPO_DE_MOVERSE` cada vez. ¿Pero cómo podremos hacer esto? `TIEMPO_DE_MOVERSE` es una constante y no pueden cambiarse constantes. Sinembargo nosotros necesitamos una variable. Pueden definirse variables fácilmente en NQC. Puedes tener hasta 32 de éstas variables, y puedes dar a cada una de ellas un nombre distinto. Aquí esta el programa espiral.

```
#define TIEMPO_DE_DARVUELTA   85

int TIEMPO_DE_MOVERSE;         // define la variable

task main()
{
  TIEMPO_DE_MOVERSE = 20;      // pon el valor inicial
  repeat(50)
  {
  OnFwd(OUT_A+OUT_C);
  Wait(TIEMPO_DE_MOVERSE);  // usa la variable para esperar
  OnRev(OUT_C);
  Wait(TIEMPO_DE_DARVUELTA);TIEMPO_DE_MOVERSE    +=5;//incrementa
la variable
  }
  Off(OUT_A+OUT_C);
}
```

Las líneas en verde indican con los comentarios. Primero nosotros definimos una variable tecleando la palabra clave **int** seguida por un nombre que nosotros escogemos. (Normalmente los programadores usan minusculas para las variables y mayúsculas para las constantes, pero esto no es necesario.) El nombre debe empezar con una letra pero puede contener digitos y giones. Ningún otro símbolo se permite. (Lo mismo pasa para las constantes, nombres de la tarea, etc.) La palabra **int** esta dada para enteros. Es decir pueden guardarse sólo números enteros en ella. Para la segunda linea de interes nosotros asignamos el valor de 20 a la variable. De este momento en adelante, cada vez que uses la variable en el programa, esta simbolizara 20. Ahora sigue el repite en el cual nosotros usamos la variable para indicar el tiempo para esperar y, al final de el repite nosotros aumentamos el valor de la variable con 5. Asi que la primera vez el robot espera 20 ticks, la segunda vez 25, la tercera vez 30, etc...

Además de agregar valores a una variable nosotros podemos multiplicar también una variable por un numero usando * =, restarla usando - = y dividiendola usando / =. (Nota que para la división el resultado se redondea al numero entero más cercano.) También puedes agregar una variable a otra, y ejecutar expresiones más complicadas. Aquí están algunos ejemplos:

```
int aaa;
int bbb, ccc;

task main()
{
  aaa = 10;
  bbb = 20 * 5;
  ccc = bbb;
  ccc /= aaa;
  ccc -= 5;
  aaa = 10 * (ccc + 3);   // aaa es ahora igual a 80
}
```

Nota en las primeras dos líneas que nosotros podemos definir multiples variables en una sola línea. Nosotros también podríamos combinar las tres en una línea.

Números al azar

En todos los programas anteriores nosotros hemos definido exactamente lo que el robot haria. Pero las cosas se ponen mucho más interesantes cuando el robot va a hacer movimientos que nosotros no sabemos. Nosotros queremos que el robot se mueva aleatoriamente. En NQC puedes crear números del azar. El programa siguiente usa esto para permitir el robot manejarse de una manera al azar (es decir que no podemos predecir como va a moverse). Constantemente avanza por una cantidad de tiempo no conocida y entonces hace un giro al azar.

```
int TIEMPO_DE_MOVERSE, TIEMPO_DE_DARVUELTA;

task main()
{
  while(true)
  {
    TIEMPO_DE_MOVERSE = Random(60);
    TIEMPO_DE_DARVUELTA = Random(40);
    OnFwd(OUT_A+OUT_C);
    Wait(TIEMPO_DE_MOVERSE);
    OnRev(OUT_A);
    Wait(TIEMPO_DE_DARVUELTA);
  }
}
```

El programa define dos variables, y entonces les asigna números al azar. `Random(60)` significa un número al azar entre 0 y 60 (también puede ser 0 o 60). Cada tiempo los números serán diferentes. (Nota que nosotros pudiéramos evitar el uso de las variables escribiéndo `Wait(Random(60))`.)

También podemos ver que hay un nuevo tipo de rizo (loop) aquí. Ya que en vez de usar la declaracion repeat nosotros escribimos while(true). La declaración while (mientras) repite los commandos debajo con tal de que la condición entre los paréntesis sea verdad. La palabra especial True (verdadero) es siempre cierta para este programa, por lo que se repiten las declaraciones entre los corchetes para siempre, así como nosotros queremos. Aprenderás más sobre la declaración de while (mientras) en el Capítulo IV.

Resumen

En este capítulo aprendiste sobre el uso de variables. Las variables son muy útiles pero, debido a las restricciones de los robots, ellas son un poco limitadas. Puedes definir sólo 32 de ellos y pueden guardar sólo numeros enteros. Pero para muchos tareas de robots esto es suficientemente bueno.

También aprendiste a crear numeros al azar, de tal forma que puedes dar conducta imprevisible al robot. Finalmente nosotros vimos el uso de la declaración while (mientras) para hacer un rizo (loop) infinito en el que nuestro robot continua para siempre.

IV. Estructuras de control

En los capítulos anteriores nosotros vimos las declaraciones repeat (repite) y while (mientras). Estas declaraciones controlan la manera en la cual las otras declaraciones en el programa se ejecutan. estas son conocicidas como " estructuras de mando.". En este capítulo nosotros veremos algunas otras estructuras del mando.

La declaración if

A veces quieres que una parte en particular de tu programa sólo se ejecute en ciertas situaciones. En este caso se usa la declaración if (si..) . Permíteme dar un ejemplo. Nosotros cambiaremos de nuevo el programa con el que hemos estado trabajando hasta ahora, pero con una nueva finalidad. Nosotros queremos el robot maneje lo largo de una línea recta y entonces haga un giro a la derecha o a la izquierda. Para hacer esto nosotros necesitamos usar de nuevo numeros el azar. Nosotros escogemos un número al azar entre 0 y 1, es decir, o tiene 0 o 1. Si (if) el número es 0 el giro sera a la derecha; si no es 0 el giro sera a la izquierda. Aquí esta el programa:

```
#define TIEMPO_DE_MOVERSE   100
#define TIEMPO_DE_DARVUELTA    85

task main()
{
  while(true)
  {
    OnFwd(OUT_A+OUT_C);
    Wait(TIEMPO_DE_MOVERSE);
    if (Random(1) == 0)
    {
      OnRev(OUT_C);
    }
    else
    {
      OnRev(OUT_A);
    }
    Wait(TIEMPO_DE_DARVUELTA);
  }
}
```

La declaración if (si...) se parece un poco a la declaración de while (mientras). Si (if..) la condición entre los paréntesis es verdad la parte entre los corchetes se ejecuta. Si (if) la condicion no se cumple la parte entre los corchetes después del palabra else (entonces haz...) se ejecuta. observemos mas de cerca la condición que estamos usamos. Ahi dice `Random(1) == 0`. esto significa que `Random(1)` debe ser igual a 0 para hacer la condicion verdadera. Podrías preguntarte por qué nosotros usamos == en lugar de =. La razón es distinguirlo de la declaración en donde se coloca el valor de una variable. Puedes comparar valores de maneras diferentes. Aquí están los más importantes:

==	igual a....
<	más pequeño que...
<=	más pequeño que o iguala a...
>	más grande que...
>=	más grande que o iguala a...
!=	no igual a...

Puedes combinar las condiciones usando && que significa "y", o || que significa "o.". Aquí están algunos ejemplos de condiciones:

`true`	siempre cierto
`false`	siempre falso ó nunca cierto
`ttt != 3`	verdadero cuando ttt no es igual a 3
`(ttt >= 5) && (ttt <= 10)`	verdadero cuando ttt se encuentra entre 5 y 10
`(aaa == 10) || (bbb == 10)`	verdadero si aaa o bbb (o ambos) es igual a 10

Nota que la declaracion if (si..) tiene dos partes. La parte inmediatamente después de la condición, que se ejecuta cuando la condición es verdad, y la parte después de la palabra else (entonces haz..) que se ejecuta cuando la condición es falsa. La palabra else y la parte después de esta son opcionales. Asi que puedes puedes dejarlos si no hay nada que hacer cuando la condición es falsa. En español las declaraciones if / else se traduce como:
Si (If)...la condicion es cierta ......entonces haz (Then) ....si la condicion no es cierta entonces haz (else)..... lo que sigue......

La declaración do

Aqui esta otra estructura de control, la declaración do . Tiene la forma siguiente:

```
do //hazlo.....
{
  statements; //comandos que va a ejecutar
}
while (condition); //.... mientras se sujete a la condicion
```

Las declaracions entre los corchetes después del **do** se ejecuta con tal de que la condición sea verdad. La condición **do** tiene la misma forma que la declaracion **if** descrita anteriormente. Aquí esta el ejemplo de un programa. EL robot corre al azar durante 20 segundos y entonces se detiene.

```
int TIEMPO_DE_MOVERSE, TIEMPO_DE_DARVUELTA, TIEMPO_TOTAL;

task main()
{
  TIEMPO_TOTAL = 0;
  do
  {
    TIEMPO_DE_MOVERSE = Random(100);
    TIEMPO_DE_DARVUELTA = Random(100);
    OnFwd(OUT_A+OUT_C);
    Wait(TIEMPO_DE_MOVERSE);
    OnRev(OUT_C);
    Wait(TIEMPO_DE_DARVUELTA);
TIEMPO_TOTAL    +=    TIEMPO_DE_MOVERSE;    TIEMPO_TOTAL    +=
TIEMPO_DE_DARVUELTA;
  }
  while (TIEMPO_TOTAL < 2000);
  Off(OUT_A+OUT_C);
}
```

Nota que en este ejemplo pusimos dos declaraciones en una línea. Esto se permite. Puedes poner tantas declaraciones en una línea como gustes (con tal de que haya puntos y comas entre ellas). Pero para la legibilidad del programa ésta no es a menudo una idea buena.

También nota que la declaración do (haz) casi se comporta igual que la declaración while(mientras). Pero en la declaración de while(mientras) la condición se prueba antes de ejecutar las instrucciones siguientes, mientras la declaración do (haz) la condición se prueba al final. Es decir que para la declaración while(mientras),las instrucciones podrían no ejecutarse nunca, pero para la declaracion do(haz) se ejecutan por lo menos una vez.

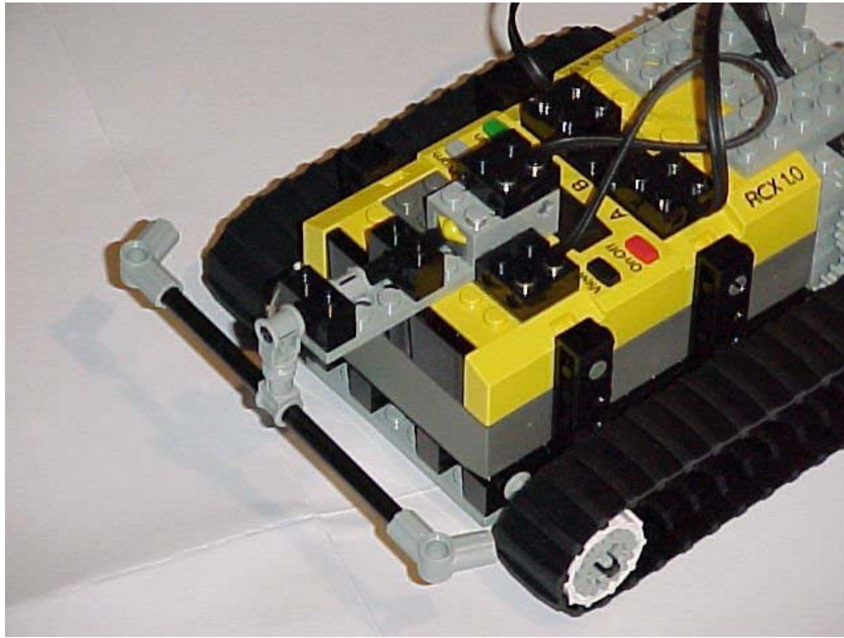
Resumen

En este capítulo nosotros hemos visto dos nuevas estructuras de control : la declaración if y la declaración do. Junto con la declaración repeat y la declaración while estas son las declaraciones que controlan la manera en la que el programa se ejecuta. Es muy importante que entiendas lo que ellas hacen. Asi que mejor juega con ellas y practica un poco más los ejemplos antes de continuar.

Nosotros también hemos visto que se pueden colocar multiples declaraciones en una linea.

V. Sensores

Uno de los aspectos buenos de los robots de Lego es que puedes conectar sensores a ellos y que puedes hacer al robot reaccionar a los sensores. Antes de que te pueda mostrar cómo hacer esto debemos cambiar el robot un poco agregando un sensor. Con este fin, construye la figura mostrada en la figura 4 en la página 28 de la constructopedia. Podrías querer hacerlo ligeramente más ancho, de tal manera que tu robot se parezca al que sigue:



Conecta el sensor a la entrada 1 en el RCX.

Esperando por un sensor

Permítenos empezar con un programa muy simple en el que el robot maneja hacia adelante hasta que pegua con algo. Aquí está:

```
task main()
{
SetSensor(SENSOR_1,SENSOR_TOUCH);//salida y tipo de sensor
OnFwd(OUT_A+OUT_C);              //enciende las salidas A y C
  until (SENSOR_1 == 1);         //hasta que sensor_1 sea = 1
  Off(OUT_A+OUT_C);              //se apagan los motores
}
```

Hay dos líneas importantes aquí. La primera línea del programa dice al robot qué tipo de sensor estamos usando. SENSOR_1 es el número de la entrada a la que nosotros conectamos el sensor. Las otras dos entradas del sensor se llaman SENSOR_2 y SENSOR_3. SENSOR_TOUCH indica que éste es un sensor del toque. Para el sensor de luz nosotros usaríamos SENSOR_LIGHT. Después de que nosotros especificamos el tipo del sensor, el programa enciende los motores y el robot empieza a moverse hacia adelante. La siguiente declaración es una instruccion muy útil. Esta espera hasta (**until**) que la condición entre los corchetes sea verdad. Esta condición dice que el valor del sensor SENSOR_1 debe ser 1 que quiere decir que el sensor se aprieta. mientras el sensor no se aprieta, el valor es 0. Asi esta declaración espera hasta que el sensor se aprieta. Entonces apagamos los motores y la tarea se termina.

Actuando en un sensor de toque

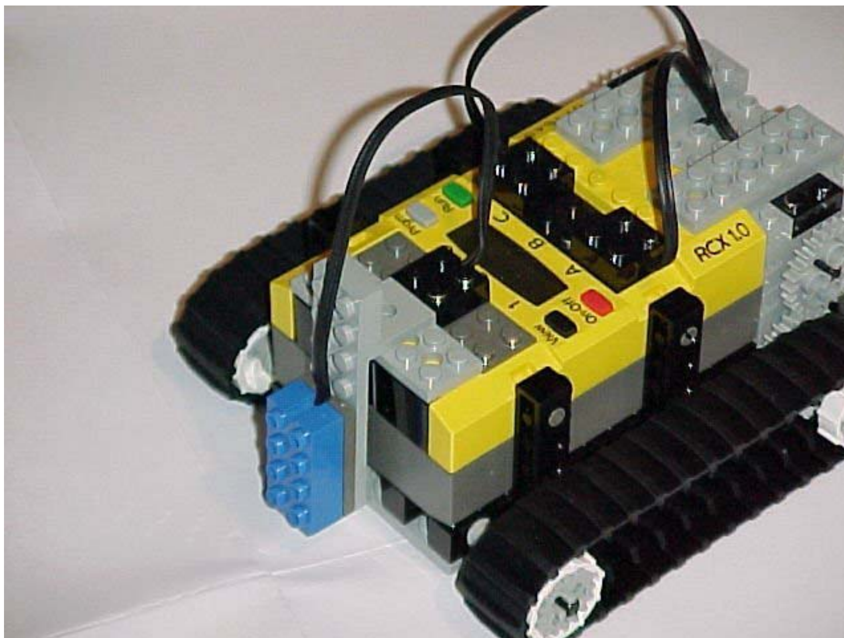
Permítenos ahora intentar hacer que el robot evite obstáculos. Siempre que el robot pegue un objeto, nosotros le permitiremos moverse un poco hacia atrás, hacer un giro, y entonces continuar avanzando. Aquí esta el programa:

```
task main()
{
  SetSensor(SENSOR_1,SENSOR_TOUCH);
  OnFwd(OUT_A+OUT_C);
  while (true)
  {
    if (SENSOR_1 == 1)
    {
      OnRev(OUT_A+OUT_C); Wait(30);
      OnFwd(OUT_A); Wait(30);
      OnFwd(OUT_A+OUT_C);
    }
  }
}
```

Como en el ejemplo anterior, nosotros indicamos primero el tipo del sensor. Luego el robot empieza moviendose hacia adelante. En el loop(rizo) infinito de la declaracion while nosotros constantemente probamos si el sensor está presionado y si esta presionado el robot se mueve hacia atrás por 1/3 de segundo, dobla a la derecha por 1/3 de un segundo, y entonces continúa hacia adelante de nuevo.

Sensores de luz

Además de los sensores de toque, también obtienes un sensor de luz conn tu sistema de MindStorms. El sensor de luz mide la cantidad de luz en una dirección particular. El sensor de luz también emite luz. De esta manera es posible apuntar el sensor de luz en una dirección particular y hacer una distinción entre la intensidad del objeto en esa dirección. Esto es en particularmente útil cuando intentamos hacer un robot que sigua una línea en el suelo. Esto es lo que nosotros vamos a hacer en el próximo ejemplo. Primero necesitamos colocar el sensor de luz al robot de tal forma que se encuentre en medio del robot, al frente, y que apunte hacia abajo. Conéctalo a entrada 2. Por ejemplo, haz una construcción como sigue:



Nosotros también necesitamos el papel de pruebas que viene con el equipo de RIS (Ese pedazo grande de papel con la linea negra en él.) La idea es ahora que el robot se asegure que la luz del sensor este sobre la linea negra. Siempre que la intensidad de la luz suba, el sensor de luz estára fuera de la linea y nosotros necesitaremos adaptar la dirección. Aquí esta un programa muy simple para esto que sólo trabajas si nosotros conducimos el robot alrededor de la linea en en el sentido de las agujas del reloj dirección.

```
#define UMBRAL 40

task main()
{
  SetSensor(SENSOR_2,SENSOR_LIGHT);
  OnFwd(OUT_A+OUT_C);
  while (true)
  {
    if (SENSOR_2 > UMBRAL)
    {
      OnRev(OUT_C);
      until (SENSOR_2 <= UMBRAL);
      OnFwd(OUT_A+OUT_C);
    }
  }
}
```

El programa indica primero que el sensor 2 es un sensor de luz. Luego hace que el robot se mueva hacia adelante y entra en un rizo(loop) infinito. Siempre que el valor de la luz sea más grande que 40 (nosotros usamos una constante aquí de tal forma que esto puede adaptarse fácilmente, por que nuestro robot depende mucho de la luz circundante) nosotros invertiremos el movimiento de un motor y esperaremos hasta que estemos de nuevo en la linea.

Como verás cuando ejecutas el programa, el movimiento no es muy fino. Intenta agregar un Wait(10) antes del until para hacer que el robot se mueva mejor. Nota que el programa no trabaja para moverse en sentido contrario a las agujas del reloj. Para habilitar el movimiento a lo largo de un camino arbitrario se requiere un programa mucho más complicado.

Resumen

En este capítulo has visto cómo trabajar con sensores de toque y sensores de luz. También hemos vimos que la orden until que es útil al usar sensores.

Yo te recomiendo que escribas varios programas en esta fase. Tienes todos los ingredientes para dar una conducta bastante complicada a tus robots. Por ejemplo, intenta poner dos sensores de toque en tu robot, uno en el frente izquierdo y el otro en el frente derecho, y haz que el robot huya de los obstáculos que pega. También, intenta hacer un robot que se quede dentro de una área indicada por una gruesa línea negra en el suelo.

VI. Tareas y subprogramas

Hasta ahora todos nuestros programas han consistido de una sola tarea. Pero los programas de NQC pueden tener tareas múltiples. También es posible colocar pedazos de código en subprogramas llamadas subrutinas las cuales pueden usarse en lugares diferentes en su programa. Usando Tareas y subprogramas tus programas seran más fácil de entender y más compactos. En este capítulo estudiaremos las distintas posibilidades.

Tareas

Un programa de NQC consiste en a lo sumo 10 tareas. Cada tarea tiene un nombre. Una tarea debe tener el nombre `main` , y esta tarea se ejecutará. Las otras tareas sólo se ejecutarán cuando alguna tarea que esta en marcha les dice que sean ejecutadas usando una orden de salida. En este momento en ambas tareas (la principal y la secundaria) se estan ejecutando simultáneamente (para que la primera tarea continúe corriendo). Una tarea también puede detener otra tarea usando la orden de la parada (stop). Después esta tarea puede reiniciarse de nuevo, pero empezará desde el principio; no del lugar donde fue detenido.

Permíteme demostrarte el uso de tareas. Pon tu sensor de toque de nuevo en tu robot. Nosotros queremos hacer un programa en el que el robot maneja alrededor en cuadrados, como antes lo hemos hecho. Pero cuando choque con un obstáculo el robot debe reaccionar . Es difícil de hacer esto en una tarea, porque el robot debe hacer dos cosas en el mismo momento: maneja (enciende y apaga los motores en los momentos correctos) y esta al tnto de los sensores. Asi que es mejor usar dos tareas para esto, una tarea que maneje los cuadrados; y la otra que reaccione a los sensores. Aquí esta el programa.

```
task main()
{
  SetSensor(SENSOR_1,SENSOR_TOUCH);
  start checa_sensores;
  start muevete_encuadros;
}

task muevete_encuadros()
{
  while (true)
  {
    OnFwd(OUT_A+OUT_C); Wait(100);
    OnRev(OUT_C); Wait(85);
  }
}

task checa_sensores()
{
  while (true)
  {
    if (SENSOR_1 == 1)
    {
      stop muevete_encuadros;
      OnRev(OUT_A+OUT_C); Wait(50);
      OnFwd(OUT_A); Wait(85);
      start muevete_encuadros;
    }
  }
}
```

La tarea principal solo pone el tipo de sensor y entonces empieza las otras tareas. Después de esto la tarea principal (llamada main) se termina. La tarea `muevete_encuadros` hace que el robot se mueva en cuadros por un tiempo infinito. La tarea `checa_sensores` verifica cuando el snsor de toque es presionado. Si es presionado entonces el programa hara lo siguiente: Primero hara que la tarea llamada `muevete_encuadros` se detenga . Esto es muy importante. una vez apagada la tarea que controla el robot en cuadros `checa_sensores` toma el control . Es aqui cuando el robot se hecha para atras un poco. Despues se puede comenzar a mover de nuevo con las instrucciones de `muevete_encuadros` para que el robot comienze a moverse en cuadros.

Es muy importante recordar que las tareas que empiezas a ejecutar estan corriendo en el mismo momento. Esto puede llevar a los resultados inesperados. El capítulo X explica estos problemas en detalle y da soluciones para ellos.

Subprogramas (Subrutinas)

A veces necesitas el mismo pedazo de código en diferentes lugares de tu programa. En este caso puedes poner el pedazo de código en un subprograma y puedes darle un nombre. Ahora puedes ejecutar este pedazo de código simplemente escribiendo su nombre dentro de la tarea. NQC (o realmente el RCX) permite a lo sumo 8 subprogramas o subrutinas. Veamos un ejemplo.

```
sub vueltas_alrededor()
{
  OnRev(OUT_C); Wait(340);
  OnFwd(OUT_A+OUT_C);
}

task main()
{
  OnFwd(OUT_A+OUT_C);
  Wait(100);
  vueltas_alrededor();
  Wait(200);
  vueltas_alrededor();
  Wait(100);
  vueltas_alrededor();
  Off(OUT_A+OUT_C);
}
```

En este programa nosotros hemos definido un subprograma que hace el robot de vueltas alrededor de su eje. La tarea principa (main task) llama el subprograma tres veces. Nota que nosotros llamamos el subprograma escribiendo su nombre y luego un parentesis . A si que parece igual que muchas de las órdenes que hemos visto.Sólo que en este caso no hay ningún parámetro, así que no hay nada entre los paréntesis.

Son necesarias algunas advertencias antes de que comienzes a trabajar con subprogramas . Los subprogramas son un poquitin raros. Por ejemplo, no pueden llamarse subprogramas desde otros subprogramas. Pueden llamarse subprogramas desde tareas diferentes pero esto no se recomienda ya que esto nos conduciria a problemas porque el mismo subprograma realmente podría correrse dos veces, en el mismo momento y por tareas diferentes y esto produce efectos indeseados. También, al llamar un subprograma desde diferentes tareas , debido a una limitación en el firmware de RCX, ya no puedes usar expresiones complicadas. Asi que ,a menos de que sepas excactamente que estas haciendo, *¡no llames un sub programa de diferentes tareas!*

Funciones inline

Como se indicó anteriormente, los subprogramas causan ciertos problemas. La parte buena es que ellos sólo se guardan una vez en el RCX. Esto ahorra memoria y, debid a que el RCX no tiene tanta memoria libre, esto es útil. Pero cuando los subprogramas son cortos es mejor utilizar funciones inline (en linea). Éstos no se guardan separadamente pero se copian a cada lugar en que se usan. Esto cuesta más memoria pero problemas como el de usar expresiones complicadas, ya está no se presentará . Ademas no hay ningún límite en el número de funciones inline.

Definir y llamar funciones inline se hace exactamente de la misma manera que usamos para los subprogramas. Sólo que en esta caso se utiliza la palabra clave **Void** (nulo) lugar de **sub**. (La palabra **void** (nulo) se usa porque esta misma palabra aparece en otros lenguajes de programacion como C.) El ejemplo anterior, pero usando funciones inline , sera de la siguiente forma:

```
void vueltas_alrededor()
{
  OnRev(OUT_C); Wait(340);
  OnFwd(OUT_A+OUT_C);
}

task main()
{
  OnFwd(OUT_A+OUT_C);
  Wait(100);
  vueltas_alrededor();
  Wait(200);
  vueltas_alrededor();
  Wait(100);
  vueltas_alrededor();
  Off(OUT_A+OUT_C);
}
```

Las funciones Inline tienen otra ventaja mas sobre los subprogramas. Pueden tener argumentos. Pueden usarse argumentos para pasar un valor para ciertas variables a una funcion inline. Por ejemplo, basandonos en el ejemplo anterior, nosotros podemos hacer que el tiempo se convierta un argumento de la función, como en el ejemplo siguiente:

```
void vueltas_alrededor(int turntime)
{
  OnRev(OUT_C); Wait(turntime);
  OnFwd(OUT_A+OUT_C);
}

task main()
{
  OnFwd(OUT_A+OUT_C);
  Wait(100);
  vueltas_alrededor(200);
  Wait(200);
  vueltas_alrededor(50);
  Wait(100);
  vueltas_alrededor(300);
  Off(OUT_A+OUT_C);
}
```

Nota que en el paréntesis despues del nombre de la función inline nosotros especificamos el argumento(s) de la función. En este caso nosotros indicamos que el argumento es un entero (hay algunas otras opciones) y que su nombre es turntime. Cuando hay más argumentos, debes separarlos con comas.

Definiendo macros

Todavia hay otra manera de dar un nombre a pedazos pequeños de código. Puedes definir macros en NQC (No te confundas con las macros de RCX Comand Center). Hemos visto antes que nosotros podemos definir constantes y podemos usar #define, siempre y cuando les demos un nombre. Pero realmente nosotros podemos definir cualquier pedazo de código. Aquí esta el mismo programa de nuevo que usa una macro para darse la vuelta.

```
#define                                             vueltas_alrededor
OnRev(OUT_C);Wait(340);OnFwd(OUT_A+OUT_C);

task main()
{
  OnFwd(OUT_A+OUT_C);
  Wait(100);
  vueltas_alrededor;
  Wait(200);
  vueltas_alrededor;
  Wait(100);
  vueltas_alrededor;
  Off(OUT_A+OUT_C);
}
```

Después de la declaración #define se escribe la palabra `vueltas_alrededor` y luego el sub programa que requieras. Ahora dondequiera que tecleas `vueltas_alrededor`, el codigo es reemplazado por este texto. Nota que el codigo y texto deben estar en una línea. (Hay maneras de poner una declaración de #define en líneas múltiples, pero esto no se recomienda.)

Las declaraciones #define son realmente mucho más poderosas. Estas también pueden tener argumentos. Por ejemplo, nosotros podemos poner el tiempo en el que el robot dara vuelta como un argumento en la declaración. Aquí es un ejemplo en el que nosotros definimos cuatro macros; una para moverse adelante, una para moverse mover al revés, una par doblar a la izquierda y una par doblar a la derecha. Cada uno tiene dos argumentos: la velocidad y el tiempo.

```
#define turn_right(s,t)   SetPower(OUT_A+OUT_C,s);OnFwd(OUT_A);OnRev(OUT_C);Wait(t);
#define turn_left(s,t)    SetPower(OUT_A+OUT_C,s);OnRev(OUT_A);OnFwd(OUT_C);Wait(t);
#define forwards(s,t)     SetPower(OUT_A+OUT_C,s);OnFwd(OUT_A+OUT_C);Wait(t);
#define backwards(s,t)    SetPower(OUT_A+OUT_C,s);OnRev(OUT_A+OUT_C);Wait(t);

task main()
{
  forwards(3,200);
  turn_left(7,85);
  forwards(7,100);
  backwards(7,200);
  forwards(7,100);
  turn_right(7,85);
  forwards(3,200);
  Off(OUT_A+OUT_C);
}
```

Es muy útil definir tales macros. Esto hace tu código más compacto y leíble. También, puedes cambiar tu código más fácilmente cuando ,por ejemplo,. cambias las conexiones a los motores.

Resumen

En este capítulo viste el uso de tareas, subprogramas, funciones inline y macros. Ellas tienen usos diferentes. Las tareas normalmente corren al mismo momento y tienen cuidado de cosas diferentes que tienen que ser hechas en el mismo momento. Los subprogramas son útiles cuando deben usarse pedazos más grandes de código en lugares diferentes de la misma tarea. Las funciones Inline son útiles cuando deben usarse pedazos de código en muchos lugares diferentes y en tareas diferentes, pero estas usan más memoria. Finalmente las macros som muy útiles para pedazos pequeños de código que deben usarse en lugares diferentes. Estas también pueden tener parámetros y pueden hacerlos aun mas útiles.

Ahora que has trabajado a través de los capítulosy has llegado hasta aquí, tienes todo el conocimiento que necesitas para que tu robot haga cosas complicadas. Los otros capítulos en esta guía didáctica te enseñan sobre otras cosas que son sólo importante en ciertas aplicaciones.

VII. Haciendo Musica

El RCX tiene una bocina dentro que puede hacer sonidos e incluso tocar pedazos simples de música. Esto es en particular útil cuando quieres hacer que el RCX te diga que algo está pasando. Pero también puede ser cómico tener la música de lo que hace el robot mientras corre alrededor.

Sonidos preprogramados

Hay seis sonidos preprogramados en el RCX, numerados del 0 a 5. Estos estan en el siguiente orden:

0	Key click
1	Beep beep
2	Barrido de frecuencia decreciente
3	Barrido de frecuencia creciente
4	'Buhhh ' Sonido de error
5	Barrido creciente rapido

Puedes hacer que suenen usando el commando `PlaySound()`.Este es un pequeño programa que hace que suenen todos.

```
task main()
{
  PlaySound(0); Wait(100);
  PlaySound(1); Wait(100);
  PlaySound(2); Wait(100);
  PlaySound(3); Wait(100);
  PlaySound(4); Wait(100);
  PlaySound(5); Wait(100);
}
```

Podrías preguntarte que por qué hay ordenes de espera (wait). La razón es que el commando toca el sonido no espera por él para terminar. Ejecuta la orden próxima inmediatamente. El RCX tiene un pequeño colchon (buffer) en el que puede guardar algunos sonidos pero después de un rato este buffer se llena y se pierden algunos sonidos. Esto no es tan serio para los sonidos pero es muy importante para música, como veremos debajo.

Nota que el argumento de `PlaySound()` debe ser una constane. No puedes poner una variable aqui !

Tocando musica

Para hecer musica mas interesante , NQC tiene el commando `PlayTone()`. Tiene dos argumentos. El primero es la frecuencia, y el segundo la duración (en tictacs de 1/100 de un segundo, como en el commando de espera `Wait`). Aquí esta una tabla de frecuencias útiles:

Sonido	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
G#	52	104	208	415	831	1661	3322	
G	49	98	196	392	784	1568	3136	
F#	46	92	185	370	740	1480	2960	
F	44	87	175	349	698	1397	2794	
E	41	82	165	330	659	1319	2637	
D#	39	78	156	311	622	1245	2489	
D	37	73	147	294	587	1175	2349	
C#	35	69	139	277	554	1109	2217	
C	33	65	131	262	523	1047	2093	4186
B	31	62	123	247	494	988	1976	3951
A#	29	58	117	233	466	932	1865	3729
A	28	55	110	220	440	880	1760	3520

Asi como vimos para el commando `PlaySound ()` , también aquí el RCX no espera a que la nota termine. Asi que si usas muchas notas seguidas mejor agrega comandos `Wait()` entre sonido y sonido (para hacerlos ligeramente más largos) . Aquí esta un ejemplo:

```
task main()
{
  PlayTone(262,40);  Wait(50);
  PlayTone(294,40);  Wait(50);
  PlayTone(330,40);  Wait(50);
  PlayTone(294,40);  Wait(50);
  PlayTone(262,160); Wait(200);
}
```

Puedes crear pedazos de música Muy facilmente usando el Piano que viene en el RCX Command Center.

Si quieres que el RCX toque música mientras maneja , mejor usa una tarea separada para eso. Aquí tienes un ejemplo de un programa bastante tonto donde el RCX maneja de un lado a otro y constantemente hace música.

```
task musica()
{
  while  (true)
  {
    PlayTone(262,40);  Wait(50);
    PlayTone(294,40);  Wait(50);
    PlayTone(330,40);  Wait(50);
    PlayTone(294,40);  Wait(50);
  }
}

task main()
{
  start musica;
  while(true)
  {
    OnFwd(OUT_A+OUT_C); Wait(300);
    OnRev(OUT_A+OUT_C); Wait(300);
  }
}
```

Resumen

En este capítulo aprendiste a permitir que el RCX haga sonidos y música. También viste cómo usar una tarea separada para la música.

VIII. Más sobre los motores

Hay varios commandos adicionales que puedes usar para controlar los motores más precisamente. En este capítulo nosotros los discutimos.

Deteniendo suavemente

Cuando usas el commando `Off()` , el motor se detiene inmediatamente,usando el freno. En NQC es también posible detener los motores de una manera más suave y sin usar el freno. Para hacer esto se utiliza el comando `Float()`. A veces esto es mejor para la tarea del robot. Aquí esta un ejemplo. Primero el robotse detiene de usano los frenos; depues se detiene sin usar los frenos. Nota la diferencia. (Realmente la diferencia es muy pequeña para este robot en particular. Pero representa una diferencia grande para algunos otros robots.)

```
task main()
{
  OnFwd(OUT_A+OUT_C);
  Wait(200);
  Off(OUT_A+OUT_C);
  Wait(100);
  OnFwd(OUT_A+OUT_C);
  Wait(200);
  Float(OUT_A+OUT_C);
}
```

Comandos avanzados

El commando `OnFwd()` de hecho hace dos cosas:enciende los motores y hace que estos den vuelta hacia adelante. El commando `OnRev()` tambien hace dos cosas : enciende los motores y hace que estos giren hacia atras. NQC tiene comandos que hacen cada accion por separado.si lo que quieres es cambiar una de las dos cosas, es mas eficiente usar comandos seperados ; esto utiliza menos memoria en el RCX, es mas rapido, y puede dar como resultado movimientos mas suaves y continuos. Los dos comandos separados son `SetDirection()` que indica la direccion (`OUT_FWD`, `OUT_REV` or `OUT_TOGGLE` el cual cambia la direccion del motor) y `SetOutput()` que enciende o apaga los motores (`OUT_ON`, `OUT_OFF` or `OUT_FLOAT`). Aqui esta un programa simple en el cual hace que el robot maneje hacia adelante, maneje hacia atras y luego maneje hacia adelante y atras de nuevo.

```
task main()
{
  SetPower(OUT_A+OUT_C,7);
  SetDirection(OUT_A+OUT_C,OUT_FWD);
  SetOutput(OUT_A+OUT_C,OUT_ON);
  Wait(200);
  SetDirection(OUT_A+OUT_C,OUT_REV);
  Wait(200);
  SetDirection(OUT_A+OUT_C,OUT_TOGGLE);
  Wait(200);
  SetOutput(OUT_A+OUT_C,OUT_FLOAT);
}
```

Nota que, al inicio de cada programa, todos los motores se colocan en direccion hacia adelante y la velocidad se coloca en 7. Pero en el ejemplo anterior, los primeros comandos ya no son necesarios.

Hay muchas otras órdenes para los motores que son atajos para las combinaciones de las órdenes que revisamos anteriormente. Aquí esta una lista completa:

`On('motores')`	Enciede los motores
`Off('motores')`	Apaga los motores
`Float('motores')`	Apaga los motores sin el freno
`Fwd('motores')`	Coloca los motores hacia adelante (Pero no los pone en marcha)
`Rev('motores')`	Coloca los motores hacia atras (Pero no los pone en marcha)
`Toggle('motores')`	Cambia la direccion de los motores en marcha
`OnFwd('motores')`	Coloca los motores hacia adelante y los pone en marcha

`OnRev('motores')`	Coloca los motores hacia adelante y los pone en marcha
`OnFor('motores','ticks')`	Enciende los motores por un tiempo dado en ticks (100 ticks = 1seg)
`SetOutput('motores','mode')`	Coloca el modo de arranque (`OUT_ON`, `OUT_OFF` o `OUT_FLOAT`)
`SetDirection('motores','dir')`	Coloca el modo de direccion (`OUT_FWD`, `OUT_REV` or `OUT_TOGGLE`)
`SetPower('motores','poder')`	Coloca el poder de salida (0-9)

Variando la Velocidad del motor

Como probablemente has notado y cambiar la velocidad de los motores no tiene mucho efecto. La razón es que estás cambiando el torque, no la velocidad. Verás sólo un efecto cuando el motor tiene una carga pesada. E incluso entonces, la diferencia entre 2 y 7 es muy pequeña. Si quieres tener mejores efectos el truco es encender y apagar los motores en sucecion rapida . Aquí esta un programa que hace esto. Este tiene solamente una tarea,llamada `motor_enmarcha` que pone en marcha los motores. Esta tarea constantemente verifica la variable `velocidad` para ver cual es la velocidad. El signo positivo(+) dara marcha hacia adelante,y el negativo (-) dara marcha hacia atras. Esto pone los motores en la dirección correcta y entonces espera por algún tiempo , tiempo que depende de la velocidad , antes de apagar los motores de nuevo. La tarea principal simplemente coloca las velocidades y espera.

```
int velocidad, __velocidad;

task motor_enmarcha()
{
  while (true)
  {
    __velocidad = velocidad;
    if (__velocidad > 0) {OnFwd(OUT_A+OUT_C);}
    if (__velocidad < 0) {OnRev(OUT_A+OUT_C); __velocidad = -
__velocidad;}
    Wait(__velocidad);
    Off(OUT_A+OUT_C);
  }
}

task main()
{
  velocidad = 0;
  start motor_enmarcha;
  velocidad = 1;   Wait(200);
  velocidad = -10; Wait(200);
  velocidad = 5;   Wait(200);
  velocidad = -2;  Wait(200);
  stop motor_enmarcha;
  Off(OUT_A+OUT_C);
}
```

Este programa puede hacerse mas poderoso, permitiendo mas rotaciones, y tambien es posible incorporar un tiempo (`wait`) despues del comando `Off()`. Experimenta con ello.

Summary

En este capitulo has aprendidosobre los comandos extra que estan disponibles para los motores: `Float()` que detiene el motor sin usar el freno, `SetDirection()` que coloca la direccion del motro(`OUT_FWD`, `OUT_REV` u `OUT_TOGGLE` que cambia la direccion del motor en marcha) y `SetOutput()` Que coloca el modo de salida del motor (`OUT_ON`, `OUT_OFF` u `OUT_FLOAT`). Viste la lista completa de comandos para el motor que estan disponibles en NQC. También aprendiste un truco para controlar la velocidad de motor de una mejor manera.

IX. Mas sobre los sensores

En el capítulo V discutimos los aspectos básicos de usar los sensores. Pero puedes hacer mucho más con los sensores. En este capítulo discutiremos la diferencia entre el modo del sensor y el tipo del sensor, veremos cómo utilizar el sensor de rotación (un tipo de sensor que no se proporciona con el RIS pero se puede comprar por separado y es muy útil), veremos algunos trucos para utilizar más de tres sensores y como hacer un sensor de proximidad.

Tipo y modo del sensor.

El comando `SetSensor()` que vimos anteriormente hace de hecho dos cosas: coloca el tipo de sensor que usaremos, y coloca el modo en el que el sensor opera. Poniendo el modo y tipo de un sensor separadamente, puedes controlar la conducta de el sensor más precisamente , esto es útil para ciertas aplicaciones particulares.

El tipo de sensor se coloca con el comando `SetSensorType()`. Hay cuatro tipos diferentes de sensores: `SENSOR_TYPE_TOUCH`, Para el sensor de toque, `SENSOR_TYPE_LIGHT`, para el sensor de luz, `SENSOR_TYPE_TEMPERATURE`, Para el sensor de temperatura (Este tipo de sensor no es parte del RIS pero puede comprarse por separado), y `SENSOR_TYPE_ROTATION`,para el sensor de rotacion (Tampoco este es parte del RIS pero se compra por separado). Colocar el tipo de sensor es particularmente importante para indicar si el sensor necesita poder (como por ejemplo la luz para el sensor de luz). No conozco de ninguna utilidad el colocar un sensor con los comandos a un tipo diferente del que realmente es.

El modo en el que el sensor trabaja se coloca con el comando `SetSensorMode()`. Existen ocho modos diferentes. `SENSOR_MODE_RAW`. En este modo, el valor que consigues cuando verificasel sensor es un número entre 0 y 1023. Ese es el valor en bruto producido por el sensor. Es decir que los numeros dependen realmente del sensor. Por ejemplo, para un sensor de toque, cuando el sensor no se empuja el valor está cerca de 1023. Cuando se empuja totalmente, está cerca de 50. Cuando se empuja parcialmente el valor va entre 50 y 1000. A si que si pusieras un sensor de toque en modo bruto (raw_mode) realmente puedes averiguar si está parcialmente presionado. Cuando el sensor es un sensor de luz, el valor va de aproximadamente 300 (mucha luz) a 800 (muy oscuro). Esto da un valor mucho mas preciso que si solo utilizaras el comando `SetSensor()`.

El segundo mod en el que puedes colocar un sensor se llama `SENSOR_MODE_BOOL`. En estew modo el valor que se le da es 0 o 1. Cuando el valor en bruto (raw_mode) por arriba de 550 el valor es 0, por debajo de 550 el valor sera 1. `SENSOR_MODE_BOOL` es el modo predeterminado para un sensor de toque. Los modos `SENSOR_MODE_CELSIUS` y `SENSOR_MODE_FAHRENHEIT` son utiles unicamente para sensores de temperatura y dan la temperatura en la escala indicada (celcius o farenheit). `SENSOR_MODE_PERCENT` Cambia el valor en bruto en un valoir entre 0 y 100 . Cada valor en bruto de 400 o menor a 400 es llevado a 100 cien por ciento. Si el valor en bruto de pronto es mas alto ,el porcentaje baja suavemente hasta 0. `SENSOR_MODE_PERCENT` Es el modo predeterminado para un sensor de luz. `SENSOR_MODE_ROTATION` Parece ser util unicamenete para el sensor de rotacion (ver mas adelante).

Hay otros dos modo interesantes: `SENSOR_MODE_EDGE` y `SENSOR_MODE_PULSE`. Estos cuentan transiciones, es decir , cuentan cambios de un valor en bruto alto a un valor en bruto bajo y viceversa. Por ejemplo cuando presionas un sensor de toque esto causa una transicion desde un valor muy alto hasta un valor muy bajo . Cuando lo dejas de presionar la transicion se da en la otra direccion (de un valor bajo a uno muy alto). Cuando colocas el sensor en el modo `SENSOR_MODE_PULSE`, solamente las transiciones de el valor bajo hacia el alto son contadas (por ejemplo cuando el sensor deja de presionarse).Asi que cada vez que el sensor es presionado y soltado el robot contara una transicion .Cuando colocas el sensor en el modo `SENSOR_MODE_EDGE`,las dos tranciciones son contadas. A si que el robot contara dos trancisiones. Con esto tu puedes saber que tan frecuentemente es pulsado el sensor.Si utilizas este comando en combinacion con un sensor de luz ppodras contar que tantas veces un lampara (muy brillante) es encendida y apagada.Or you can use it in combination. Por cierto , Cuando estas contando eventos,deberias colocar el contador con un valor de 0. Para esto tu ulizaras el comando `ClearSensor()`. Esto limpia el contador para el o los sensores indicados.

Permítenos mirar un ejemplo. El programa siguiente usa un sensor de toque para dirigir el robot. Conecta el sensor del toque con un alambre largo a la entrada uno. Si tocas el sensor rápidamente dos veces el robot se movera hacia adelante. Si lo tocas una vez se detiene el movimiento .

```
task main()
{
  SetSensorType(SENSOR_1,SENSOR_TYPE_TOUCH);
  SetSensorMode(SENSOR_1,SENSOR_MODE_PULSE);
  while(true)
  {
    ClearSensor(SENSOR_1);
    until (SENSOR_1 >0);
    Wait(100);
    if (SENSOR_1 == 1) {Off(OUT_A+OUT_C);}
    if (SENSOR_1 == 2) {OnFwd(OUT_A+OUT_C);}
  }
}
```

Nota que nosotros pusimos el tipo del sensor primero y depues el modo en el que trabajara. Parece que esto es esencial porque cambiar el tipo de sensor tambien afectara el modo en el que trabaje.

El sensor de rotación

El sensor de rotación es un tipo muy útil de sensor que desgraciadamente no es parte del RIS normal. Sin embargo puede comprarse separadamente. El sensor de rotación contiene un agujero a través de cual puedes poner un eje. El sensor de rotación mide la cantidad de rotacion de el eje. Una rotación completa del eje es de 16 pasos (o -16 si lo ruedas en reversa). Los sensores de rotación son muy útiles para hacer que el robot haga movimientos precisos y controlados. Puedes hacer que un eje se mueva la cantidad exacta. Si necesitas un control mas fino que solo 16 pasos, siempre puedes usar engranes para conectarlo a un eje que se mueva más rápidamente, y use ese engranaje para contar los pasos.

Una aplicación normal es tener dos sensores de rotación conectados a las dos ruedas del robot que se controlan con dos motores. Para un movimiento recto ambas ruedas deben con una velocidad igual. Desgraciadamente, los motores normalmente no corren a exactamente la misma velocidad. Usando los sensores de la rotación tú pueden ver que una de las ruedas es más rápida. Puedes detener ese motor por un momento (es mejor si utilizas `Float()`) Hasta que los sensores tengan ese valor de nuevo. El programa siguiente hace esto. Permite que el robot se ponga ne marcha sobre una línea recta. Para usarlo, cambia tu robot conectando los dos sensores de rotación a las dos ruedas. Conecta los sensores a la entrada 1 y 3.

```
task main()
{
  SetSensor(SENSOR_1,SENSOR_ROTATION); ClearSensor(SENSOR_1);
  SetSensor(SENSOR_3,SENSOR_ROTATION); ClearSensor(SENSOR_3);
  while (true)
  {
    if (SENSOR_1 < SENSOR_3)
      {OnFwd(OUT_A); Float(OUT_C);}
    else if (SENSOR_1 > SENSOR_3)
      {OnFwd(OUT_C); Float(OUT_A);}
    else
      {OnFwd(OUT_A+OUT_C);}
  }
}
```

El programa indica primero que ambos sensores son sensores de rotación, y restablece los valores a cero. Luego empieza una rutina infinita. En la rutina nosotros verificamos si las dos lecturas del sensor son iguales. Si los valores son iguales el robot se mueve hacia adelante. Si uno es más grande, el motor correcto se detiene hasta que ambas lecturas sean iguales.

Claramente éste es un programa muy simple. Puedes extender este para hacer que el robot maneje distancias exactas, o para permitirle hacer giros muy precisos.

Colocando sensores múltiples en una entrada

El RCX tiene sólo tres entradas para que puedas conectar sólo tres sensores a él. Cuando quieres hacer robots más complicados (y compraste algunos sensores extras) estos no podrían ser bastante para ti. Afortunadamente, con algunos trucos, puedes conectar dos (o más aun) sensores a una entrada.

Lo más fácil es conectar dos sensores de toque a una entrada. Si uno de ellos (o ambos) está presionado, el valor es 1,si no esta presionado el valor es 0 . No puedes distinguir los dos pero a veces esto no es necesario. Por ejemplo, cuando pones un sensor de toque al frente y uno en la parte de atrás del robot, sabes cuál estápresionado basadoen la dirección en la que el robot está manejando. Pero también puedes poner el modo de la entrada en bruto (raw) (ve anteriormente). Ahora puedes conseguir mucho más información. Si tienes suerte, el valor cuando el sensor se aprieta no es el mismo para ambos sensores. Si éste es el caso realmente puedes distinguir entre los dos sensores. Y cuando los dos se aprietan consigues un valor muy más bajo (alrededor de 30) as que tambien puedes detectar esto.

También puedes conectar un sensor de toque y un sensor lde luz a una entrada. Coloca el tipo para sensor de luz (de lo contrario el sensor de luz no trabajará). Pon el modo en bruto (raw). En este caso, cuando el sensor de toque se presiona consigues un valor en bruto debajo de 100. Si no se empuja consigues el valor del sensor de luz el cual nunca está debajo de 100. El programa siguiente usa esta idea. El robot debe equiparse con un sensorde luz que apunte hacia abajo, y un parachoques al frente conectado a un sensor de toque. Conecta los dos de ellos a entrada 1. El robot manejará al azar alrededor dentro de una área iluminada. Cuando el sensor de luz ve una línea oscura (valor en bruto > 750) manejara en reversa un poco. Cuando el sensor de toque choca con algo (valor en bruto debajo de 100) hace lo mismo. Aquí esta el programa:

```
int ttt,tt2;

task moverandom()
{
  while (true)
  {
    ttt = Random(50) + 40;
    tt2 = Random(1);
    if (tt2 > 0)
      { OnRev(OUT_A); OnFwd(OUT_C); Wait(ttt); }
    else
      { OnRev(OUT_C); OnFwd(OUT_A);Wait(ttt); }
    ttt = Random(150) + 50;
    OnFwd(OUT_A+OUT_C);Wait(ttt);
  }
}

task main()
{
  start moverandom;
  SetSensorType(SENSOR_1,SENSOR_TYPE_LIGHT);
  SetSensorMode(SENSOR_1,SENSOR_MODE_RAW);
  while (true)
  {
    if ((SENSOR_1 < 100) || (SENSOR_1 > 750))
    {
      stop moverandom;
      OnRev(OUT_A+OUT_C);Wait(30);
      start moverandom;
    }
  }
}
```

Yo espero que el programa esté claro. Hay dos tareas. La tarea moverandom hace que el robot se mueva al azar. La tarea principal empieza moverandom primero, coloca el sensor y entonces espera por algun evento externo. Si el valor del sensor que esta leyendo se pone demasiado bajo (presionado) o demasiado alto (fuera del área blanca) detiene el movimiento al azar, da marcha hacia atras un poco, y empieza a moverse al azar de nuevo.

También es posible conectar dos sensores de luz a la misma entrada. El valor en bruto está de alguna manera relacionada a la cantidad combinada de luz recibida por los dos sensores. Pero esto es bastante incierto y parece dificil de usar. Conectar otros sensores con rotación o los sensores de temperatura parecen no ser útil.

Haciendo un sensor de proximidad

Al usar sensores de toque, tu robot puede reaccionar cuando pega con algo. Pero sería mucho mejor si el robot pudiera reaccionar antes de que pegue con algo. Deberia saber que está cerca de algún obstáculo. Desgraciadamente no hay ningún sensor disponible para esta tarea. Pero hay un truco que nosotros podemos usar para esto. El robot tiene un puerto infra-rojo con el que se puede comunicar con la computadora, o con otros robots. (Nosotros veremos más sobre la comunicación entre los robots en Capítulo XI.) Resulta que el sensor de luz que viene con el robot es muy sensible a luz infra-roja. Nosotros podemos construir un sensor de proximidad basado en esto. La idea es que una tarea mande mensajes infra-rojos. Otra tarea mide fluctuaciones en la intensidad de luz que se refleja de los objetos. Entre mas alta este la fluctuación, más cerca estaremos a un objeto.

Para usar esta idea, pon el sensor de luz sobre el puerto infra-rojo en el robot, apuntando adelante. De esta manera él sólo medira el reflejo de luz infra-roja. Conéctalo a entrada 2. Nosotros acostumbramos usar el modo en bruto para el sensor de luz para ver las fluctuaciones tan preciso como posible. Aquí esta un programa simple que permite al robot caminar hacia adelante hasta que se encuentre cerca de un objeto y entonces hara un 90 giro del grado al derecho.

```
int ultimonivel;            // Para guardar el valor previo

task envia_señal()
{
  while(true)
    {SendMessage(0); Wait(10);}
}

task checa_señal()
{
  while(true)
  {
    ultimonivel = SENSOR_2;
    if(SENSOR_2 > ultimonivel + 200)
      {OnRev(OUT_C); Wait(85); OnFwd(OUT_A+OUT_C);}
  }
}

task main()
{
  SetSensorType(SENSOR_2, SENSOR_TYPE_LIGHT);
  SetSensorMode(SENSOR_2, SENSOR_MODE_RAW);
  OnFwd(OUT_A+OUT_C);
  start envia_señal;
  start checa_señal;
}
```

La tarea `envia_señal` envia 10 señales IR cada segundos, usando el comando `SendMessage(0)`. La tarea checa_señnal constantemente guarda el valor que le envia el sensor de la luz. Entonces verifica si (ligeramente después) el valor es por lo menos de 200 unidades más alto e ndicada una fluctuación grande. En ese caso, permite al robot hacer un giro de 90 grados ala derecha. El valor de 200 es bastante arbitrario. Si lo haces más pequeño, los giros del robot seran mucho antes del chocar contra el obstaculo frenta a el. Si lo haces más grande, se acerca mas a ellos. Pero esto también depende del tipo de material y la cantidad de luz disponible en el cuarto. Debes experimentar o debes usar mecanismo más diestro para encontrar el valor correcto.

Una desventaja de la técnica es que sólo funciona en una dirección. Por lo que probablemente aun necesites los sensopres de toque a los lados para evitar colisiones. Pero la técnica es muy útil para robots que deben manejar en laberintos. Otra desventaja es que que no te puedes comunicar de la computadora al robot porque esto interferirá con los comandos infra-rojos enviados por el robot. (Puede ser que el control remoto de tu televisión tampoco funcione.)

Resumen

En este capítulo nosotros hemos visto varios problemas adicionales sobre los sensores. Nosotros vimos cómo colocar separadamente el tipo y modo de un sensor y cómo esto pudiera usarse para conseguir mas información. Nosotros aprendimos a usar el sensor de rotación. Y nosotros vimos cómo pueden conectarse sensores múltiples

a una entrada del RCX. Finalmente, nosotros vimos un truco para usarla conexion infraroja del robot en combinación con un sensorde luz, para crear un sensor de proximidad. Todos estos trucos son sumamente útiles al construir robots más complicados. Los sensores siempre juegan allí un papel crucial.

X. Tareas paralelas

As has been indicated before, tasks in NQC are executed simultaneously, or in parallel as people usually say. This is extremely useful. In enables you to watch sensors in one task while another task moves the robot around, and yet another task plays some music. But parallel tasks can also cause problems. One task can interfere with another.

Un mal programa.

Considera el programa siguiente. Aquí una tarea maneja el robot alrededor en cuadrados (como ya hemos visto ntes) y la segunda tarea checa el sensor de toque. Cuando el sensor está presionado,se mueve un poco hacia atras, y hace un giro de 90 grados.

```
task main()
{
  SetSensor(SENSOR_1,SENSOR_TOUCH);
  start checa_sensores;
  while (true)
  {
    OnFwd(OUT_A+OUT_C); Wait(100);
    OnRev(OUT_C); Wait(85);
  }
}

task checa_sensores()
{
  while (true)
  {
    if (SENSOR_1 == 1)
    {
      OnRev(OUT_A+OUT_C);
      Wait(50);
      OnFwd(OUT_A);
      Wait(85);
      OnFwd(OUT_C);
    }
  }
}
```

Esto se parece un programa absolutamente válido. Pero si lo ejecutas encontrarás un poco de conducta inesperada. Intenta lo siguiente: Haz que el robot toque algo mientras está volviéndose. Empezará a hecharse en reversa , pero inmediatamente se movera hacia adelante de nuevo y pegara con el obstáculo. La razón para esto es que las tareas pueden interferir. Lo siguiente está pasando. El robot está doblando a la derecha, es decir, la primera tarea está en su segunda declaración de espera . Ahora el robot pega el sensor. Empieza yendo al revés, pero en ese mismo momento, la tarea principal está lista con espera y mueve el robot hacia adelante; hacia el obstáculo. La segunda tarea está en espera en este momento por lo que no notará la colisión. Ésta claramente no es la conducta que nos gustaría ver. El problema es que, mientras la segunda tarea está esparando nosotros no nos dimos cuenta que la primera tarea todavía estaba corriendo, y que sus acciones interfieren con las acciones de la segunda tarea.

Deteniendo y reiniciando tareas

Una manera de resolver este problema es asegurarse que en cualquier momento sólo una tarea está manejando el robot. Éste era el acercamiento que nosotros observamos en el Capítulo VI. Permíteme repetir el programa aquí.

```
task main()
{
  SetSensor(SENSOR_1,SENSOR_TOUCH);
  start checa_sensores;
  start muevete_encuadros;
}

task muevete_encuadros()
{
  while (true)
  {
    OnFwd(OUT_A+OUT_C); Wait(100);
    OnRev(OUT_C); Wait(85);
  }
}

task checa_sensores()
{
  while (true)
  {
    if (SENSOR_1 == 1)
    {
      stop muevete_encuadros;
      OnRev(OUT_A+OUT_C); Wait(50);
      OnFwd(OUT_A); Wait(85);
      start muevete_encuadros;
    }
  }
}
```

La tarea checa_sensores sólo mueve el robot después de detener la tarea de muevete_encuadros. Asi es que esta tarea no puede interferir en el movimiento para evadir el obstaculo. Una vez el procedimiento de respaldo se termina, empieza muevete_encuadros de nuevo.

Aunque ésta es una solución buena para el problema anterior, hay un problema. Cuando nosotros reiniciamos muevete_encuadros, empieza de nuevo al principio. Esto está bien para nuestra pequeña tarea, pero a menudo ésta no es la conducta requerida. Nosotros preferiríamos detener la tarea donde se encuentra y despues continúa a paritr del punto en donde se encuentra. Desgraciadamente esto no puede hacerse fácilmente.

Usando Semaforos

Una técnica normal para resolver este problema es usar una variable para indicar qué tarea está en mando de los motores. Las otras tareas no tienen permiso de manejar los motores hasta que la primera tarea indica , usando la variable, que está listo. Semejante variable se llama a menudo un semáforo. Veamos uno de estos semáforos. Nosotros asumimos que un valor de 0 indica que ninguna tarea está dirigiendo los motores. Ahora, siempre que una tarea quiera hacer algo con los motores ejecutara los comandos siguientes:

```
until (sem == 0);
sem = 1;
//Haz algo con los motores
sem = 0;
```

Asi que primero esperamos mientras nadie necesite los motores. Entonces nosotros exigimos el mando poniendo sem = 1. Ahora nosotros podemos controlar los motores. Cuando terminemos pondremos sem de regreso a su valor de 0. Aquí encontraras que el programa siguiente, fue hecho usando un semáforo. Cuando el sensor de toque toca algo, el semáforo se coloca y el procedimiento de respaldo es realizado. Durante este procedimiento la tarea muevete_encuadros debe esperar. En este momento el respaldo está listo, el semáforo se pone en 0 y muevete_encuadros puede continuar.

```
int sem;

task main()
{
  sem = 0;
  start muevete_encuadros;
  SetSensor(SENSOR_1,SENSOR_TOUCH);
  while (true)
  {
    if (SENSOR_1 == 1)
    {
      until (sem == 0); sem = 1;
      OnRev(OUT_A+OUT_C); Wait(50);
      OnFwd(OUT_A); Wait(85);
      sem = 0;
    }
  }
}

task muevete_encuadros()
{
  while (true)
  {
    until (sem == 0); sem = 1;
    OnFwd(OUT_A+OUT_C);
    sem = 0;
    Wait(100);
    until (sem == 0); sem = 1;
    OnRev(OUT_C);
    sem = 0;
    Wait(85);
  }
}
```

Podrías defender que no es necesario en muevete_encuadros poner el semáforo a 1 y de nuevo a 0. Todavía esto es útil. La razon es que el comando `OnFwd()` son en realidad dos comandos (ve el capitulo **¡Error! No se encuentra el origen de la referencia.**). y no quieres que esta tarea sea interumpida por la otra.
Los semáforos son muy útiles y, cuando estás escribiendo programas complicados con tareas paralelas, ellos casi siempre son necesarios. (Hay todavía una ligera oportunidad que pudiera fallar. Intenta deducir por qué.)

Resumen

En este capítulo nosotros estudiamos algunos de los problemas que pueden ocurrir cuando usas tareas diferentes. Siempre ten mucho cuidado con efectos colaterales. La conducta inesperada es debida a estos. Nosotros vimos dos maneras diferentes de resolver tales problemas. La primera solución detiene y reinicia tareas para asegurarse que sólo una tarea crítica está corriendo en cada momento. El segundo acercamiento usa semáforos para controlar la ejecución de tareas. Esto garantiza que en cada momento que sólo la parte crítica de una tarea se ejecuta.

XI. Communication between robots

Si posees más de un RCX que este capítulo es para ti. Los robots pueden comunicar entre sí a través del puerto infra-rojo. Usando esto puedes tener robots múltiples colaboranod o luchando entre ellos. También puedes construir un robot grande que use dos RCX, de tal formal que puedes tener seis motores y seis sensores (o mas aun usando los descritos en el Capítulo IX).

La comunicación en tre robots en general trabaja de la siguiente forma. Un robot puede usar el comando `SendMessage()` para enviar un valor entre 0-255)hacia el púerto infrarojo. Todos los demas Robots reeciben este mensaje y lo guardan. El programa en un robot puede preguntar por el valor del ultimo mensaje enviado usando `Message()`. Basa do en este valor el robot podra hacer que el robot inicie ciertads tareas asignadas a este numero.

Dando Ordenes

A menudo, cuando tienes dos o más robots, uno es el líder. Nosotros lo llamamos el amo. Los otros robots son esclavos. El robot amo envía comandos a los esclavos y a los esclavos ejecutan estos. A veces los esclavos podrían enviarle informaciónde vuelta al amo, por ejemplo el valor de un sensor. Por lo que que necesitas escribir dos programas, uno para el amo y uno para el escalvo(s). De ahora en adelante nosotros asumiremos que tenemos solamente un esclavo. Permítenos empezar con un ejemplo muy simple. Aquí el esclavo puede realizar tres órdenes diferentes: muevete adelante, muevete al revés, y detente. Este programa consiste en un rizo (loop) simple. En este rizo se coloca el valor de 0 al mensaje en curso usando el comando `ClearMessage()`. Despues espera hasta que el mensaje se vuelve diferente de 0. Basado en el valor del mensaje recibido se ejecuta uno de las tres ordenes. Aquí esta el programa.

```
task main()             // ESCLAVO
{
  while (true)
  {
    ClearMessage();
    until (Message() != 0);
    if (Message() == 1) {OnFwd(OUT_A+OUT_C);}
    if (Message() == 2) {OnRev(OUT_A+OUT_C);}
    if (Message() == 3) {Off(OUT_A+OUT_C);}
  }
}
```

El amo tiene un programa aun más simple. Simplemente envía los mensajes que corresponden a las órdenes y entonces espera un momento. En el programa debajo este ordena al esclavo moverse hacia adelante,después de dos segundos, se mueve en reversa, y de nuevo, después de dos segundos, detenerse.

```
task main()             // MASTER
{
  SendMessage(1); Wait(200);
  SendMessage(2); Wait(200);
  SendMessage(3);
}
```

Después de que has escrito estos dos programa, necesitas transmitirlos a los robots. Cada programa debe ir a uno de los robots. Asegúrate que apagaste uno (también ve las precauciones extra mas adelante). Ahora enciende los dos robots y empieza los programas: primero el de el esclavo y despues el de el amo.

Si tienes muchos esclavos, tienes que transmitir el programa del esclavo a cada uno de ellos (no simultáneamente; lee mas adelante). Ahora todos los esclavos realizarán las mismas acciones exactamente.

Para permitir que los robots se comuniquen entre sí definimos, lo que se llama, un protocolo: Nosotros decidimos que un 1 signifique "muevanse adelante", un 2 "muevanse hacia atras", y un 3 "detenganse". Es muy importante definir tales protocolos cuidadosamente, en particular cuando se involucran muchas comunicaciones. Por ejemplo, cuando hay más esclavos, podrías definir un protocolo en el que dos números son envíados (con un pequeño intervalo de tiempo entre ellos): el primer número es el número del esclavo, y el segundo es la orden

real. El esclavo que tenga ese primer número sólo el realizara la acción cuando se envie su numero. (Esto requiere que cada esclavo tienga su propio número que puede ser logrado permitiendo a cada esclavo tener un programa ligeramente diferente. Como por ejemplo si le colocas una constante diferente.)

Elegiendo a un líder

Como vimos anteriormente, al tratar con robots múltiples, cada robot debe tener su propio programa. Sería muy más fácil si nosotros pudieramos transmitir simplemente un programa a todos los robots. Pero entonces la pregunta es: ¿quién es el amo? La respuesta es fácil: permite que los robots decidan. Permíteles elegir a un líder que los otros seguirán. ¿Pero cómo hacemos esto? La idea es bastante simple. Nosotros permitimos que cada robot espere una cantidad al azar de tiempo y entonces enviamos un mensaje. El que envíe un mensaje primero es el líder. Este esquema podría fallar si dos robots esperan exactamente que la misma cantidad de tiempo pero esto es bastante improbable. (Puedes construir esquemas más complicados que detecten esto y prueben una segunda elección en semejante caso.) Aquí esta el programa que lo hace:

```
task main()
{
  ClearMessage();
  Wait(200);              //estan encendidos todos los robots?
  Wait(Random(400));      // espera entre 4 y 0 segundos
  if (Message() > 0)      // alguien fue el primero
  {
    start slave;
  }
  else
  {
    SendMessage(1);       // ahora yo soy el amo
    Wait(400);            //asegurate de que los demas lo sepan
    start master;
  }
}

task master()
{
  SendMessage(1); Wait(200);
  SendMessage(2); Wait(200);
  SendMessage(3);
}

task slave()
{
  while (true)
  {
    ClearMessage();
    until (Message() != 0);
    if (Message() == 1) {OnFwd(OUT_A+OUT_C);}
    if (Message() == 2) {OnRev(OUT_A+OUT_C);}
    if (Message() == 3) {Off(OUT_A+OUT_C);}
  }
}
```

Transmite este programa a todos los robots (uno por uno, no en el mismo momento; lee mas adelante). Inicia los robots aproximadamante en el mismo momento y ve lo que pasa. Uno de ellos debe tomar orden y el otro(s) debe seguir las órdenes. En ocasiones raras, ninguno de ellos se vuelve el líder. Como indicó anteriormente, esto exige protocolos más cuidadosos que resolver.

Precauciones

Tienes que ser un poco cuidadoso al tratar con robots múltiples. Hay dos problemas: Si dos robots (o un robot y la computadora) envían infromacion al mismo tiempo esta información podría perderse. El segundo problema es que, cuando la computadora envía un programa al mismo tiempo a los muchos robots, esto causa problemas.

Permítenos empezar con el segundo problema. Cuando transmites un programa al robot, el robot le dice a la computadora si recibe correctamente (partes de) el programa. La computadora reacciona aa eso enviando nuevos pedazos o reenviando los mismos pedazos. Cuando dos robots estan encendidos, los dos empezarán diciéndole a

la computadora si reciben el programa correctamente. La computadora no entiende esto (no sabe que hay dos robots!). Como resultado, las cosas salen mal y el programa se corrompe. Los robots no harán las cosas correctas. *¡Siempre asegúrate que, mientras estás transmitiendo programas, sólo un robot esta encendido!*

El otro problema es que sólo un robot puede enviar un mensaje en cualquier momento. Si dos mensajes están siendo envíados bruscamente al mismo momento, ellos podrían perderse. Tampoco, un robot puede enviar y puede recibir mensajes en el mismo momento. Éste no es ningún problema cuando sólo un robot envía mensajes (hay sólo un amo) pero por otra parte podría ser un problema serio. Por ejemplo, puedes imaginar escribir un programa en el que un esclavo envía un mensaje cuando tropieza con algo, de tal forma que el amo pueda tomar acción. Pero si el amo envía una orden en el mismo momento, el mensaje se perderá. Para resolver esto, es importante definir tu protocolo de comunicación tal que, en caso de que una comunicación falle, esto se corrija. Por ejemplo, cuando el amo envía una orden, debe recibir una respuesta del esclavo. Si no recibe una respúesta en un tiempo determinado, él amo reenviara la orden. Esto produciría un pedazo de código que se parece a este:

```
do
{
  SendMessage(1);
  ClearMessage();
  Wait(10);
}
while (Message() != 255);
```

Aquí se usa 255 para el reconocimiento.

A veces, cuando estás tratándo con muchos robots, podrías querer eso sólo un robot que esta muy cerca de reciba la señal. Esto puede ser logrado añadiendo el commando `SetTxPower(TX_POWER_LO)` al programa del amo. En este caso las señales IR enviadas son muy bajas y sólo un robot cerca y defrente al amo podran "oir". Esto es en particular útil al construir un robot más grande de dos RCX. Usa `SetTxPower(TX_POWER_HI)` para colocar al robot de nuevo a un modo de transmision de alto rango.

Resumen

En este capítulo estudiamos algunos de los aspectos básicos de comunicación entre los robots. La comunicación usa loa órdenes para enviar, limpiar, y checar mensajes. Nosotros vimos que es importante definir un protocolo para definir como trabajaran las comunicaciones. Tales protocolos juegan un papel crucial en cualquier forma de comunicación entre las computadoras. Nosotros también vimos que hay varias restricciones en la comunicación entre robots que hacen aun mas importante definir buenos protocolos.

XII. Más órdenes

NQC tiene varias órdenes adicionales. En este capítulo nosotros discutiremos tres tipos: el uso de cronómetros, órdenes para controlar la pantalla, y el uso del rasgo de datalog del RCX.

Cronómetros

El RCX tiene cuatro cronómetros preconstruidos. Estos cronómetros hacen tictac en incrementos de 1/10 de segundo. Los cronometros estan numerados de 0 a 3. Puedes volver a iniciar un cronometro con el comando `ClearTimer()` a y obtener el valor del cronometro en ese momento con `Timer()`. Aquí esta un ejemplo de cómo utilizar un cronometro. El siguiente programa permite al robot manejar al azar por un periodo de 20 segundos.

```
task main()
{
  ClearTimer(0);
  do
  {
    OnFwd(OUT_A+OUT_C);
    Wait(Random(100));
    OnRev(OUT_C);
    Wait(Random(100));
  }
  while (Timer(0)<200);
  Off(OUT_A+OUT_C);
}
```

Tal vez quieras comparar este programa con el que se reviso en el capitulo IV que hace exactamente la misma tarea . Pero al usar cronometros es definitivamente mas simpler.

Los cornometro son muy utiles para replazar el comando `Wait()`. Puedes poner al robot a "dormir" por una cantidad determinada de tiempo al iniciar el cronometro y esperar hasta que alcanze un valor en particular. Pero también puedes reaccionar en otros eventos (por ejemplo. de los sensores) mientras estas esperando. El programa siguiente es un ejemplo de esto. Permite al robot manejar hasta que hayan pasado 10 segundos, o el sensor de toque choque con algo.

```
task main()
{
  SetSensor(SENSOR_1,SENSOR_TOUCH);
  ClearTimer(3);
  OnFwd(OUT_A+OUT_C);
  until ((SENSOR_1 == 1) || (Timer(3) >100));
  Off(OUT_A+OUT_C);
}
```

No te olvides que los cronómetros trabajan en tictaces de 1/10 de un segundo, mientras que. el comando de espera (`Wait`) usa tictaces de 1/100 de un segundo.

La pantalla.

Es posible controlar la pantalla del RCX de dos maneras diferentes. En primer lugar, puedes indicar qué desplegar: el reloj del sistema, uno de los sensores, o uno los motores. Esto es equivalente a usar el botón de vista negro en el RCX. Para colocar el tipo de pantalla , usa al comando `SelectDisplay()`. El programa siguiente muestra todas las siete posibilidades, uno después de la otra.

```
task main()
{
  SelectDisplay(DISPLAY_SENSOR_1); Wait(100);  // Entrada 1
  SelectDisplay(DISPLAY_SENSOR_2); Wait(100);  // Entrada 2
  SelectDisplay(DISPLAY_SENSOR_3); Wait(100);  // Entrada 3
  SelectDisplay(DISPLAY_OUT_A);  Wait(100);  // Salida A
  SelectDisplay(DISPLAY_OUT_B);  Wait(100);  // Salida B
  SelectDisplay(DISPLAY_OUT_C);  Wait(100);  // Salida C
  SelectDisplay(DISPLAY_WATCH);  Wait(100);// Reloj de sistema
}
```

Nota que no se deberia usar `SelectDisplay(SENSOR_1)`.

La segunda manera que puedes controlar la pantalla es controlando el valor del reloj del sistema. Puedes esto para desplegar ,por ejemplo, Información de diagnóstico. Para esto usa el comando `SetWatch()`. Aquí esta un programa diminuto que usa esto:

```
task main()
{
  SetWatch(1,1); Wait(100);
  SetWatch(2,4); Wait(100);
  SetWatch(3,9); Wait(100);
  SetWatch(4,16); Wait(100);
  SetWatch(5,25); Wait(100);
}
```

Nota quelos argumentos para `SetWatch()` deben ser constantes.

Datalogging

El RCX puede guardar valores de variables, lecturas del sensor, y cronómetros, en un pedazo de memoria lammado datalog. Los valores en el datalog dentro del RCX no pueden usarse, pero pueden ser leídos por tu computadora. Esto es útil ,por ejemplo,. para checar qué está pasando en tu robot. RCX Command Center tiene una ventana especial en la que puedes ver los contenidos actuales del datalog.

Usar el datalog consiste de tres pasos: Primero, El programa en NQC debe definir el tamaño de el datalog usando el comando `CreateDatalog()`. Esto también limpia los contenidos actuales del datalog. Segundo, los valores pueden ser escritos en el datalog usando al comando `AddToDatalog()`. Los valores se escribirán uno después el otro. (Si miras la pantalla del RCX que verás ese uno después el otro, ya que aparecen cuatro partes de un disco. Cuando el disco está completo, el datalog está lleno.) Si el final del datalog se alcanza, no pasa nada. Ya no se guardan nuevos valores. El tercer paso es subir el datalog a tu PC. Para esto, escoge en RCX Command Center el commando Datalog en el menú de Herramientas. Luego aprieta el boton etiquetado como **Upload Datalog**, y todo los valores aparecen. Puedes mirarlos o puedes salvarlos a un archivo para hacer algo más con ellos. Las personas que programan en NQC han acostumbrado este rasgo para,por ejemplo,. haz un escáner con el RCX.

Aquí esta un ejemplo simple de un robot con un sensor de luz. El robot maneja durante 10 segundos, y el valor se escribe en el datalog cinco veces cada segundo.

```
task main()
{
  SetSensor(SENSOR_2,SENSOR_LIGHT);
  OnFwd(OUT_A+OUT_C);
  CreateDatalog(50);
  repeat (50)
  {
    AddToDatalog(SENSOR_2);
    Wait(20);
  }
  Off(OUT_A+OUT_C);
}
```

XIII. NQC referencia rápida

En este capitulo se encuentra una lista de todas las construcciones de las declaraciónes, commandos, constantes, etc.,. para NQC. La mayoría de éstas se han tratado en los capítulos anteriores,asi que solo se dan descripciones cortas.

Declaraciones

Declaracion	Descripcion
while (*cond*) *cuerpo*	Ejecuta el *cuerpo* cero o mas veces mientras la condicion sea cierta.
do *cuerpo* while (*cond*)	Ejecuta *cuerpo* una o mas veces mientras la condicione se cumpla.
until (*cond*) *cuerpo*	Ejecuta *cuerpo* cero a mas veces hasta que la condicion sea cierta.
break	Rompe desde el *cuerpo* de while/do/until
continue	Brinca hasta la siguiente iteracion de while/do/until *cuerpo*
repeat (*expression*) *cuerpo*	Repite *cuerpo* un numero especificado de veces
if (*cond*) *stmt1* **if** (*cond*) *stmt1* **else** *stmt2*	Ejecuta *stmt1* si la condicion es cierta. Ejecuta *stmt2* (cuando este presente) si la condicion es falsa.
start *task_name*	Inicia la tarea especificada.
stop *task_name*	Detiene la tarea especificada.
function(*args*)	Llama una funcion usando los argumentos que se han dado.
var = *expression*	Evalua una expresion y la asigna a la variable.
var += *expression*	Evalua una expresion y la suma a la variable.
var -= *expression*	Evalua una expresion y la resta a la variable
var *= *expression*	Evalua una expresion y la multiplica por la variable.
var /= *expression*	Evaluna una expresion y la divide entre la variable
var |= *expression*	Evalua la expresion y le asigna el operador OR a la variable
var &= *expression*	Evalua la expresion y le asigna el operador AND a la variable
return	Devuelve de la función hacia el llamador
expression	Evalua la expresion

Condiciones

Se usan condiciones dentro de las declaraciones de control para tomar decisiones. En la mayoría de los casos, la condición involucrará una comparación entre las expresiones.

Condition	Meaning
true	siempre verdadero
false	siempre falso
expr1 == expr2	prueba si las expresiones son iguales
expr1 != expr2	prueba si las expresiones no son iguales
expr1 < expr2	prueba si una expresión es menor que otra
expr1 <= expr2	prueba si una expresión es menor o igual a otro
expr1 > expr2	prueba si una expresión es mayor que otra
expr1 >= expr2	prueba si una expresión es mayor que o igual a otra
! condicion	negación lógica de una condición
cond1 && cond2	operador logico AND de dos condiciones (verdadero si y sólo si ambas condiciones son verdad)
cond1 || cond2	operador lógico OR para dos condiciones (verdadero si y sólo si por lo menos una de las condiciones es verdad)

Expresiones

Hay un numero de valores diferentes que pueden usarse dentro de las expresiones incluyendo las constantes, variables, y valores del sensor. Nota que SENSOR_1, SENSOR_2, y SENSOR_3 son macros que se expanden en SensorValue(0), SensorValue(1), y SensorValue(2) respectivamante.

Valor	Descripcion
number	Valor de una constante (ej. "123")
variable	El nombre de una variable (ej. "x")

Timer(*n*)	Valor *n* del cronometro, cuando *n* esta entre 0 y 3.
Random(*n*)	Numero al azar entre 0 y *n*
SensorValue(*n*)	Valor actual del sensor *n*, cuando esta entre 0 y 2
Watch()	Valor del reloj del sistema.
Message()	Valor del ultimo mensaje IR recibido

Pueden combinarse valores usando operadores. Algunos de los operadores sólo pueden usarse evaluando expresiones constantes lo que siginifica que los opernados tambien deben ser constantes, o expresiones que involucran unicamente constantes. Los operadores se listan aquí en orden decreciente (del más alto a más bajo).

Operador	Descrition	Associatividad	Restriccion	Ejemplo
abs() **sign**()	Valor Absoluto Signo del operando	n/a n/a		abs(x) sign(x)
++ --	Incremento Decremento	Izquierda Izquierda	solo variables solo variables	x++ or ++x x-- or --x
- ~	menos Unario negacion Bitwise (unaria)	derecha derecha	solo constantes	-x ~123
* / %	Multiplicación División Modulo	izquierda izquierda izquierda	sólo constantes	x * y x / y 123 % 4
+ -	Suma Substracción	izquierda izquierda		x + y x - y
<< >>	Cambio izquierdo Cambio derecho	izquierda izquierda	sólo constantes sólo constantes	123 << 4 123 >> 4
&	Bitwise AND	izquierda		x & y
^	Bitwise XOR	izquierda	sólo constantes	123 ^ 4
|	Bitwise OR	izquierda		x | y
&&	AND Lógico	izquierda	sólo constantes	123 && 4
||	OR lógico	izquierda	sólo constantes	123 || 4

Funciones RCX

La mayoría de las funciones exigen a todos los argumentos ser expresiones constantes (número u operaciones que involucren otras expresiones constantes). Las excepciones son funciones que usan un sensor como un argumento, y aquéllas que pueden usar cualquier expresión. En el caso de sensores, el argumento deberia ser el nombre del sensor: SENSOR_1, SENSOR_2, o SENSOR_3. En algunos casos son nombres predefinidos (ej. SENSOR_TOUCH) para las constantes apropiadas.

Function	Description	Example
SetSensor(*sensor, config*)	Configura sensor.	SetSensor(SENSOR_1, SENSOR_TOUCH)
SetSensorMode(*sensor, modo*)	Coloca el modo del sensor	SetSensor(SENSOR_2, SENSOR_MODE_PERCENT)
SetSensorType(*sensor, tipo*)	Coloca el tipo de sensor	SetSensor(SENSOR_2, SENSOR_TYPE_LIGHT)
ClearSensor(*sensor*)	Limpia el valor del sensor	ClearSensor(SENSOR_3)
On(*salidas*)	Enciende una o mas salidas.	On(OUT_A + OUT_B)
Off(*salidas*)	Apaga una o mas salidas.	Off(OUT_C)
Float(*salidas*)	Deja que las salidas "floten"	Float(OUT_B)
Fwd(*salidas*)	Define la direccion hacia delante.	Fwd(OUT_A)
Rev(*salidas*)	Define la direccion hacia atrás.	Rev(OUT_B)
Toggle(*salidas*)	Cambia la direccion de las salidas.	Toggle(OUT_C)
OnFwd(*salidas*)	Se enciende en direccion	OnFwd(OUT_A)

	hacia delante	
`OnRev(Salidas)`	Se enciende en direccion hacia atrás	`OnRev(OUT_B)`
`OnFor(Salidas,tiempo)`	Se enciende por un numero especificado de cantecimas de segundo. El tiempo puede ser una expresion.	`OnFor(OUT_A, 200)`
`SetOutput(salidas, modo)`	Coloca el modo de salida.	`SetOutput(OUT_A, OUT_ON)`
`SetDirection(salidas, dir)`	Coloca la direccion de salida.	`SetDirection(OUT_A, OUT_FWD)`
`SetPower(salidas, poder)`	Coloca el poder de salida. Puede ser una expresion.	`SetPower(OUT_A, 6)`
`Wait(tiempo)`	espera por una cantidad de tiempo en centecimas de segundo. El tiempo puede ser una expresion.	`Wait(x)`
`PlaySound(sonido)`	Toca el sonido especificado(0-5).	`PlaySound(SOUND_CLICK)`
`PlayTone(freq, duracion)`	Toca un tono de la frecuencia especificada por unac antidad definida de tiempo (en decimas de secundo)	`PlayTone(440, 5)`
`ClearTimer(cronometro)`	Coloca el cronometro (0-3) a n valor de 0	`ClearTimer(0)`
`StopAllTasks()`	Detiene todas las tareas que se esten ejevcutando.	`StopAllTasks()`
`SelectDisplay(modo)`	selecciona uno de 7 modos de pantalla 0: reloj del sistema, 1-3: valor del sensor, 4-6: modo de salida. El mod puede ser una expresion.	`SelectDisplay(1)`
`SendMessage(mensage)`	Envia un mensaje IR (1-255). El mensaje puede ser na expresion.	`SendMessage(x)`
`ClearMessage()`	Limpia el Buffer de mensajes IR.	`ClearMessage()`
`CreateDatalog(tamaño)`	Crea un nuevo datalog del tamaño especificado.	`CreateDatalog(100)`
`AddToDatalog(valor)`	Añade un valor al datalog. el valor puede ser una expresion.	`AddToDatalog(Timer(0))`
`SetWatch(houras, minutos)`	Coloca el valor de el reloj del sistema.	`SetWatch(1,30)`
`SetTxPower(hi_lo)`	Coloca el poder de transimision de el IR en alto o bajo.	`SetTxPower(TX_POWER_LO)`

Constantes de RCX

Muchos de los valores para las funciones de RCX se han como nombrado constantes que pueden ayudar hacen código más leíble. Donde sea posible, usa una constante nombrada en lugar de un valor en bruto.

Configurraciones del sensor para SetSensor()	`SENSOR_TOUCH, SENSOR_LIGHT, SENSOR_ROTATION, SENSOR_CELSIUS, SENSOR_FAHRENHEIT, SENSOR_PULSE, SENSOR_EDGE`
Modos para SetSensorMode()	`SENSOR_MODE_RAW, SENSOR_MODE_BOOL, SENSOR_MODE_EDGE, SENSOR_MODE_PULSE, SENSOR_MODE_PERCENT, SENSOR_MODE_CELSIUS, SENSOR_MODE_FAHRENHEIT,`

	SENSOR_MODE_ROTATION
Tipos para SetSensorType()	SENSOR_TYPE_TOUCH, SENSOR_TYPE_TEMPERATURE, SENSOR_TYPE_LIGHT, SENSOR_TYPE_ROTATION
Salidas para On(), Off(), etc.	OUT_A, OUT_B, OUT_C
Modos para SetOutput()	OUT_ON, OUT_OFF, OUT_FLOAT
Direcciones para SetDirection()	OUT_FWD, OUT_REV, OUT_TOGGLE
Poder de salida para SetPower()	OUT_LOW, OUT_HALF, OUT_FULL
Sonidos para PlaySound()	SOUND_CLICK, SOUND_DOUBLE_BEEP, SOUND_DOWN, SOUND_UP, SOUND_LOW_BEEP, SOUND_FAST_UP
Modos para SelectDisplay()	DISPLAY_WATCH, DISPLAY_SENSOR_1, DISPLAY_SENSOR_2, DISPLAY_SENSOR_3, DISPLAY_OUT_A, DISPLAY_OUT_B, DISPLAY_OUT_C
Nivel de poder Tx para TxPower()	TX_POWER_LO, TX_POWER_HI

Palabras Clave

Son esas palabras reservadas por el compilador de NQC para el lenguaje de programacion. Es un error usar cualquiera de éstas para nombres de funciones, tareas, o variables. Las siguientes son las palabras clavet: **__sensor, abs, asm, break, const, continue, do, else, false, if, inline, int, repeat, return, sign, start, stop, sub, task, true, void, while.**

XIV. Comentarios finales

Si has trabajado a tu manera durante esta guía didáctica ahora puedes considerarte un experto en NQC. Si no has hecho esto ahora , es tiempo para empezar a experimentar. Con creatividad en diseño y programarcion puedes hacer que los robots de Lego hagan las cosas más maravillosas.

Esta guía didáctica no cubrió todos los aspectos del RCXCommand Center. Te recomiendo que leas la documentación en alguna fase. También NQC todavía está en desarrollo. La versión futura podría incorporar funcionalidad adicional. No se trataron muchos conceptos de la programación en esta guía didáctica. En particular, nosotros no consideramos aprender conducta de robots ni otros aspectos de inteligencia artificial.

También es posible dirigir un robot de Lego directamente de una PC. Esto te exige escribir un programa en un idioma como Visual Básico, Java o Delphi. También es posible permitir a ese programa trabajar junto con un programa de NQC corriendo en el propio RCX. Semejante combinación es muy poderosa. Si estás interesado en esta manera de programar tu robot, mejor empieza con referencia técnica de el espíritu (spirit.ocx) de la pagina de Lego el MindStorms.

http://www.legomindstorms.com/

Internet es una fuente perfecta para información adicional. Algunos otros puntos de arranque importantes están en los eslabones de el autor de esta guia didactica:

http://www.cs.uu.nl/people/markov/lego/

y LUGNET, el LEGO® Users Group Network (no oficial):

http://www.lugnet.com/

También puede encontrarse mucha información en el newsgroup lugnet.robotics y lugnet.robotics.rcx.nqc en lugnet.com.

Traducido al español por Roberto Hernandez Reyes "robolabs" Terminado el 17 de julio 2K

robolesca@yahoo.com
robolabx@yahoo.com
http://roberto.go2click.com

¡¡¡¡ Gracias a **Mark Overmars** por escribir este documento tan NECSESARIO para los fanaticos de lego mindstorms !!!!