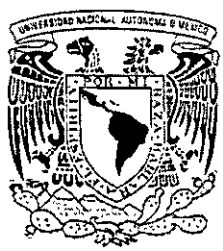
ROBOT LEGO NXT PROGRAMADO CON ROBOTC

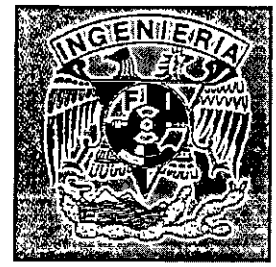
CA-102

Del 4 al 6 de Septiembre de 2008

Prof. C. Rafael Augusto Sobrevilla Figueroa

APUNTES GENERALES

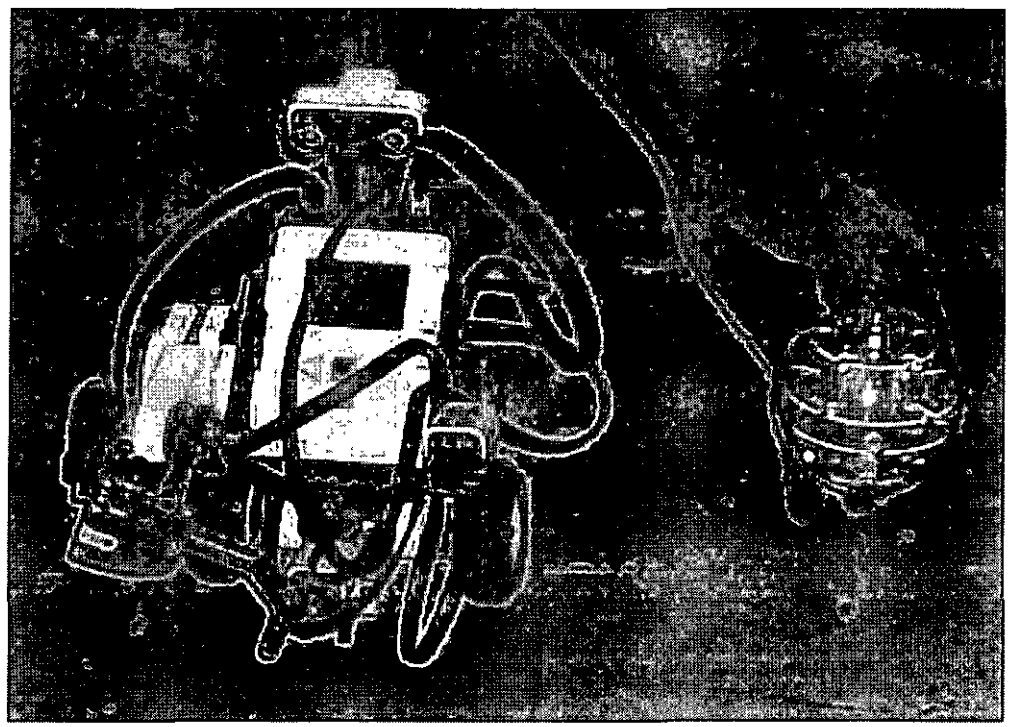




UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

ROBOT LEGO NXT PROGRAMADO CON ROBOTC



Prof. C. Rafael Augusto Sobrevilla Figueroa

4 de septiembre del 2008, V1.01

Curso: ROBOT LEGO NXT programado con RobotC

OBJETIVO

Aprender a programar en el lenguaje RobotC, y poder observar los resultados de nuestros programas en el kit LEGO NXT. Se mostrará la potencialidad de estas dos herramientas para crear aplicaciones en robótica, monitoreo de experimentos y control de procesos.

TEMARIO

1. LENGUAJE DE PROGRAMACIÓN C
1.1. Antecedentes de lenguaje C
1.2. Variables
1.3. Palabras Reservadas
1.4. Asignación
1.5. Las constantes
1.6. Los números enteros
1.7. Booleanos
1.8. Operadores
1.9. Control de flujo de condicionales
1.10. Sentencias que implican ciclos
1.11. La sentencia "mientras que" (while)
1.12. Funciones

2. LEGO NXT
2.1. Brick del LEGO NXT
2.2. Botones del brick LEGO NXT
2.3. Altavoz del LEGO NXT
2.4. LCD del LEGO NXT
2.5. Primeros programas ejemplo.
2.6. Piezas Electrónicas
2.6.1. Actuadores y líneas de salida del LEGO NXT
2.6.1.1. Servo motores
2.6.1.2. Luces del LEGO NXT
2.6.1.3. Programas Ejemplo de actuadores
2.6.2. Sensores líneas de entrada del NXT
2.6.2.1. Interruptor
2.6.2.2. Micrófono
2.6.2.3. Infrarrojo Inactivo
2.6.2.4. Infrarrojo Activo
2.6.2.5. Sonar
2.6.2.6. Programas ejemplo de sensores
2.7. Entendiendo las piezas del LEGO NXT
2.8. Robots de aplicación con NXT.

EVALUACIÓN

50% Teoría
50% Actividades practicas

CALENDARIO
4, 5 y 6 de septiembre de 2008

HORARIO
De 8:00 a 11:00 horas.

BIBLIOGRAFÍA

C++ PROGRAMACIÓN ORIENTADA A OBJETOS
Diego G. Ruiz, MP Ediciones S.A., Primera edición
Buenos Aires, Argentina 2004.

THE UNOFFICIAL LEGO MINDSTORMS NXT INVENTOR'S GUIDE
David J. Perdue, No Starch Press Inc, Primera edición
San Francisco CA, USA 2007.

1. LENGUAJE DE PROGRAMACIÓN C

1.1. Antecedentes de lenguaje C

El lenguaje C fue desarrollado en los laboratorios Bell durante la década del 70. Las principales características del lenguaje C son:

Portabilidad
El código escrito en lenguaje C puede ser compilado para trabajar en distintas plataformas (PC, Laptops, PDA (Personal Digital Assitan), Robots, Microprocesadores, etc.). Gracias a la gran popularidad del lenguaje existen compiladores en una gran diversidad de plataformas.

Eficiencia
Si codificásemos el mismo algoritmo en distintos lenguajes de programación, el lenguaje ensamblador sería sin lugar a dudas quien generaría el programa que mejor uso de los recursos memoria y procesador; en segundo lugar estaría el lenguaje C. En pocas palabras, utilizando un lenguaje de programación de alto nivel, como el C, lograríamos una eficiencia similar a la de un lenguaje de bajo nivel, como el ensamblador. No por nada el lenguaje C es tan popular en la programación de aplicaciones que requieren hacer un uso muy eficiente de los recursos.

Primer programa en C
Después de ver esta pequeña introducción al lenguaje C escribamos un pequeño programa que nos muestre un texto en la pantalla del bloque NXT.

```
// Ejemplo 1
task main() {
        eraseDisplay();
        nxtDisplayTextLine(2,"L E G O  NXT");
                for(;;) {
                }
}
```

Analizaremos este programa línea por línea

Como todo programa en C se debe poseer una función principal de nombre main, en este caso esta antecedida por la palabra task y precedida por una llave abierta ({)

"task main()"
Esta palabra nos denota que se va ejecutar una tarea en este caso la "tarea principal", se hace de esta manera para que pueda ser identificada por el ensamblador propio del bloque del NXT.

> *No es lo mismo llamar la función **main** que **Main** o **MAIN**. El nombre correcto es con cada una de sus letras en minúscula. Para C no es indistinto el uso de mayúsculas y minúsculas, como sucede en otros lenguajes de programación. Hay que prestar atención a este aspecto.*

" { "
La llave abierta indica el inicio de un bloque de código que finaliza con una llave cerrada (}); debido a que se encuentra seguido al nombre de la función, este bloque es el contenido de la misma, es decir, su definición.

Por lo tanto, la estructura básica de un programa en C es la siguiente:

```
task main() {

/*Código, cuerpo de la función main. */

}
```

```
eraseDisplay();
```
Lego se llama a una función diseñada para borrar el display del NXT en su totalidad. Esta función esta especialmente diseñada para este compilador. El punto y coma (;) indican el final de la línea de comando. Siempre al final de una sentencia tenemos que colocarlo.

```
nxtDisplayTextLine(2,"L E G O  NXT");
```
Es una función exclusiva de este compilador que nos permite desplegar texto en la pantalla del Lego NXT, primero indicamos el número de línea de la pantalla donde se escribirá el mensaje y entre comillas el mensaje a mostrar. En este caso el número de renglón es 2, y el mensaje "L E G O NXT".

> *El display de Lego NXT dividido en 8 renglones en los que podemos escribir mensajes (el primer renglón es 0 y el ultimo es el número 7).*

```
for(;;) {

}
```

En estas líneas de código generemos un ciclo infinito para detener el flujo del programa en un punto específico. El ciclo for con todos sus parámetros lo veremos más a detalle cuando tratemos el tema: Sentencias de Control.

Comentarios

En un principio nuestros programas serán pequeños, y analizar qué sucede dentro de él será sencillo, ya que bastará con revisar el código escrito. Sin embargo, a medida que él mismo crece se torna muy conveniente poder introducir notas aclaratorias sobre qué es lo que pretendemos que el programa haga. Estas notas son muy útiles para que nuestro código sea entendido rápidamente al ser analizado por otro programador o quizás nosotros mismo releyendo el código que hemos escrito tiempo atrás. Para que el compilador del lenguaje C no interprete el texto aclaratorio como intentos fallidos de instrucciones, debemos indicar de algún modo que estamos introduciendo un comentario.

Una manera de hacerlo es anteponiendo a la secuencia de caracteres una barra asterisco (/*) al inicio del comentario y al final de la secuencia de caracteres colocar un asterisco barra (*/).

```
//Ejemplo 2
//Programa en C con comentarios ( /*  */ )
/*****************************************************************
                                Nombre del programa
Autor: Tu nombre
Fecha:

Descripcion del programa ...

*****************************************************************/

task main() {

/*      eraseDisplay();
              nxtDisplayTextLine(0,"L E G O  NXT");
                        for(;;) {

                        } */

              nxtDisplayTextLine(3,"Tercer RENGLON");

              for(;;) {

              }

}
```

Esta manera de comentar tiene la característica que permite comentar varias líneas al mismo tiempo.

```
task main () {

      /*      Un comentario de este tipo
              puede tener muchas
              lineas */
}
```

Otra manera de introducir un comentario en C es con la secuencia de caracteres barra barra (//). A diferencia del tipo visto anteriormente, la doble barra especifica que el comentario finaliza al terminar la línea (no hay secuencia de fin de comentario).

```
task main () {

        // Este es un comentario, finaliza al terminar la línea
}

//Ejemplo 3
//Programa en C con comentarios ( // )
/*****************************************************************
                                Nombre del programa
Autor: Tu nombre
Fecha:

Descripcion del programa ...

*****************************************************************/

task main() {

                eraseDisplay();
                nxtDisplayTextLine(0,"L E G O  NXT");
                                for(;;) {

                                }

//              nxtDisplayTextLine(3,"Tercer RENGLON");

//              for(;;) {
//
//              }

}
```

▲

1.2. Variables

Las variables son utilizadas para almacenar datos. Sin embargo, para esto deberemos antes declararlas. Una declaración es una sentencia que introduce un nuevo elemento dentro de un programa. En ella, deberemos especificar su tipo y su identificador.

- El tipo especifica el tamaño en memoria asignado para el dato alamecenado por la variable y el correcto modo de presentarlo.
- El identificador es el nombre que tendrá la variable y que utilizaremos para referenciarla en el programa. C especifica una regla que debe seguir todo identificado, es decir que no podemos nombrar nuestra variable del modo que se nos ocurra:
    - Debe comenzar con una letra o un carácter de guión bajo.
    - Debe continuar con una letra, un número o un carácter de guión bajo.
    - No debe coincidir con una palabra reservada.

Sintaxis:

<tipo de dato> <identificador>

Ejemplo:

```
int     contador      ;
(tipo)  (identificador)  (fin de sentencia)
```

1.3 Palabras reservadas

C reserva una serie de palabras que no pueden ser utilizadas como identificadores:

Asm	do	inline	return	typedef
Auto	double	If	short	typeid
Bool	dynamic_cast	int	signed	typename
Break	else	long	sizeof	union
Case	enum	mutable	static	unsigned
catch	Explicit	namespace	static_cast	using
Char	Export	new	struct	virtual
Class	Extern	operator	switch	void
Conts	False	private	template	volatile
conts_cast	float	protected	this	wchar_t
Continue	for	public	throw	while
Default	friend	register	Trae	
Delete	goto	reinterpret_cast	try	

Nombres de variables válidos:

A	s6	Contador
mi_variable	b_a_3_2	contadores

1.4. Asignación

La asignación es una instrucción por medio de la cual podremos modificar el valor contenido en una variable. Veamos:

```
int     a;      // Declaramos una variable de tipo antero llamada " a ".
a   =   4;      // Asignación.
```

La instrucción de asignación es:

<variable> = <expresión> ;

Ejemplo:

```
        contador            =           4           ;
      (variable)  (operador de asignación)  (expresión) (fin de sentencia)
```

También es posible declarar una variable al mismo tiempo que le asignamos un valor:

```
int a = 4;      // Declaración y asignación.
```

Ahora veamos un ejemplo de los temas estudiados:

```
// Ejemplo 4
//Programa en C con variables
task    main() {
                        int var_a = 2;
                        int c,b;
                        b = 2*var_a;
                        c = b + 1;
        nxtDisplayTextLine(0," a = %4d",var_a);
        nxtDisplayTextLine(1,"  b = %4d",b);
        nxtDisplayTextLine(2,"   c = %4d",c);

        for (;;) {
        }
}
```

En este código en las primeras líneas de la función main() se encuentra la declaración de las variables que utilizaremos en el programa. En C no es necesario que la declaración se haga en un punto determinado del programa (no existe una sección para tal fin), sólo basta con que se realice antes de referenciar la variable en cuestión. A muchos programadores les gusta declarar todas las variables que se utilizarán en una función en las primeras líneas de las mismas (tal como hicimos en el listado anterior), otros prefieren hacerlo una línea de código antes de su primera intervención. Primero se declara la variable "var_a" que va ser de tipo entero y se realiza una asignación de una constante. En la segunda línea se declaran dos variables "c,b" que son del tipo entero. En la tercera línea se realiza una asignación a la variable "b" de una expresión compuesta por una multiplicación de la variable "var_a" con una constante numérica. En la cuarta línea se realiza una asignación a la variable "c" de una expresión compuesta por la operación suma de la variable "b" con una constante numérica. En la quinta, sexta y séptima línea se muestran en la pantalla del NXT el contenido de las

variables. Por último se genera un ciclo infinito (for(;;)) para detener el flujo del programa.

> *El lenguaje de programación C no inicializa las variables, es decir las variables no comienzan en un valor conocido. Le otorga una posición de memoria pero no limpia el valor que ahí existe.*

1.5. Las constantes

Las constantes son muy similares a las variables, con la diferencia que a éstas sólo se les podrá dar un valor al momento de su declaración, luego cualquier intento de modificación será tomado como error por parte del compilador.
Las constantes se declaran del mismo modo que las variables, sólo que se deben anteponer la sentencia "const" antes del tipo:

```
//Ejemplo 5
task main() {

const int max = 20;

}
```

Tipos de datos fundamentales
Hemos visto que las constantes y las variables deben especificar un tipo. C especifica en total cinco tipos de datos fundamentales.

1.6. Los números enteros

El tipo de dato básicos de enteros especificando en C es el "int", sin embargo se aceptan diversos modificadores para especificar un entero con signo o sin signo (sólo números mayores o iguales a cero); también es posible especificar un entero corto ("short int" o solamente "short" o solamente "short") o un entero largo ("long int" o solamente "long").

Ejemplos:

```
int a;                  // variable entera con signo
short int b;            // variable entera corta con signo
short c;                // igual a la línea anterior
long int d;             // variable entera larga con signo
long e;                 // igual a la línea anterior
unsigned int f;         // variable entera sin signo
unsigned short int g;   // variable entera corta sin signo
unsigned short h;       // igual a la línea anterior
unsigned long int i;    // variable entera larga sin signo
unsigned long j;        // igual a la línea anterior
```

Ahora bien ¿Qué diferencia existe entre declarar una variable tipo entera corta, entera o entera larga? El lenguaje especifica que un tipo dato short int debe ser menor o igual en tamaño a un int, el cual deberá ser menor o igual a un tipo de dato long int; pero no indica exactamente el tamaño del dato que ocupará. En plataformas de 32 bits (386, 486, 586 Pentium) estos tipos de datos ocupan:

Tipo de dato	Límites	Memoria
short int	-32,767 a 32,767	16 bits
unsigned short int	0 a 65,535	16 bits
int	-2,147,483,647 a 2,147,483,647	16 bits
unsigned int	0 a 4,294,967,2895	32 bits
long int	-2,147,483,647 a 2,147,483,647	32 bits
unsigned long int	0 a 4,294,967,2895	32 bits

Números no enteros

C diferencia en tipo a un número entero de un número con parte fraccionaría, para estso números de datos existen dos tipos: "float" y "double".

La diferencia entre ellos es la precisión. Las computadoras se llevan muy bien con los números enteros, pero para trabajar con números decimales requieren utilizar un recurso matemático llamado "punto flotante", de esta manera es posible almacenar una cantidad no fija de decimales pero con una determinada precisión máxima. Esta precisión en definitiva se encuentra determinada por el tamaño total que ocupa la variable en memoria; C no especifica un tamaño concreto pero sí establece que la variable del tipo "float" debe ser menor o igual en tamaño al tipo "double", la cual debe ser menor o igual a la del tipo "long double".

Por lo tanto para declar una variable no entera podremos escribir el tipo como:

"float" (precisión simple) "double" (precisión doble)

Ejemplos:

```
float a;          //variable decimal de precisión simple
double b;         // variable decimal de precisión doble
long double c;    // variable decimal larga de precisión doble
```

En plataformas de 32 bits (386, 486, 586 Pentium) estos tipos de datos ocupan:

Tipo de dato	Límites	Memoria
float	10^{-38} a 10^{38}	32 bits
double	10^{-308} a 10^{308}	64 bits
long double	10^{-308} a 10^{308}	64 bits

1.9. Booleanos

Este tipo de dato bool (booleano) sólo podrá contener dos valores: **true** (verdadero) o **false** (false).

```
// Ejemplo 6
bool  pushsensor ()
{
        if (SensorValue[S1]<200)
        {
                return(true);
        }
                return(false);
}

task main()
 {
         for (;;)
        {
                if (pushsensor())
                {
                        nxtDisplayTextLine(3,"PRESIONADO");
                        wait10Msec(10);
                }
                nxtDisplayTextLine(3,"NO PRESIONADO");
        }
}
```

1.10. Operadores

Son las representaciones graficas usadas para representar las operaciones matemáticas y lógicas que podemos emplear. Al utilizar operadores podemos combinar números y variables.

```
int a = 5 + 6;  // Declaro 'a' como tipo entero y le asigno un valor igual a 7

int a = 8;
int b = a / 2; // Declaro 'b' como tipo entero y le asigno un valor igual a 4
```

Y no sólo podemos emplear números enteros en nuestras expresiones, también podemos usar tipos de datos numéricos decimales.

```
float a = 5.7f / 2;
```

También podemos combinar tipos de datos en operaciones e indicar en que tipo de dato queremos el resultado.

```
int a = 5;
float b = 4.3f;
float resultado = a * b;
```

Sin embargo, cuando operamos de este modo debemos tener cuidado ya que el tipo de dato resultante estará en función de la operación que realicemos. Por ejemplo, la suma de dos números enteros dará como resultado un número entero y la división de dos números enteros también dará como resultado un número entero, y esto último es lo que precisamente puede traer muchos problemas. ¿Cuál es el resultado de la operación 5 dividido entre 2?, sabemos que la respuesta es 2.5, pero nuestra computadora en cambio nos retorna 2, ya que la operación fue entera y el resultado debe ser entero. El colocar una variable de tipo flotante al resultado no soluciona este inconveniente:

```
float a = 5 / 2;          // El resultado de esta operación es 2
```

Para resolver este problema debemos convertir al menos uno de los operadores en flotante para que de este modo la división no sea entre números enteros sino entre un entero y un número flotante.

```
float a = 5.0f / 2;       // Ahora el resultado será 2.5
```

1.11. Control de flujo de condicionales

En los pequeños programas que hemos ejecutado, el procesador ejecutaba las líneas de código escritas una a una de modo secuencial hasta llegar al final del programa. Es posible alterar este flujo de ejecución por medio de distintas sentencias y así le podremos decir a nuestro compilador que una determinada secuencia de instrucciones no debe ejecutarse sólo una vez si no diez veces, o quizás que repita una porción de código hasta que el usuario presione una tecla determinada.

1.12. Sentencias que implican ciclos

1.12.1 SENTENCIAS CONDICIONALES

La sentencia if.

Si (expresión) entonces hacer {comandos} sino hacer {comandos}

La expresión usualmente es una comparación que utiliza operadores:

= =	igual a
!=	distinto a
<	menor a
>	mayor a
<=	menor o igual a
>=	mayor o igual a

```
// Ejemplo 7
if ( SensorValue[S1]<200 )
{
        nxtDisplayTextLine(3,"PRESIONADO");
        wait10Msec(10);
}
else
{
        nxtDisplayTextLine(3,"NO PRESIONADO");
}
```

También podemos validar dos condiciones al mismo tiempo por medio de operadores lógicos.

```
if ( a = = 1 && b = = 2 )
```

El operador && es el operador lógico AND, por lo que esta expresión fue equivalente a:

"Si 'a' es igual a 1 y 'b' es igual a 2, entonces ..."

Por lo tanto, sólo ingresar'iamos al cuerpo de la sentencia if cuando la expresión "a = = 1" y la expresión "b = = 2" se evaluaran por verdadero.
También podríamos desear realizar una determinada acción cuando a sea igual a 1 o b se igual a 2, entonces utilizaremos el operador lógico OR (||):

```
if ( a = = 1 || b = = 2 )
```

De este modo podríamos crear expresiones aún más complejas, combinando operadores lógicos a nuestro gusto. .

La sentencia switch

Existe otra sentencia que permite dividir el flujo de ejecución en función del valor de una determinada variable. En este caso, la sentencia switch es especialmente útil cuando requerimos dividir el flujo en varios caminos mutuamente excluyentes.

```
switch (expresión)
{
        case <expresión constante> : <comandos> break;
        case <expresión constante> : <comandos> break;
        case <expresión constante> : <comandos> break;
        case <expresión constante> : <comandos> break;
        default:
}
```

```
const tSensors SE2      = (tSensors) S2;

task main()
{       long vol;
        while(1)
        {
                        while(SensorValue(SE2) <= 930)
                        {
                                if (SensorValue[SE2] <= 900)
                                {       vol=1;
                                }
                                if (SensorValue[SE2] <= 870)
                                {       vol=2;
                                }
                                if (SensorValue[SE2] <= 840)
                                {       vol=3;
                                 }
                                if (SensorValue[SE2] <= 810)
                                {       vol=4;
                                 }
                                 if (SensorValue[SE2] <= 750)
                                { vol=5;
                                }
                        switch(vol)
                        {
                        case 1:
                                nxtDisplayTextLine(2,"Volumen = 1");
                                wait10Msec(10);
                                break;
                        case 2:
                                nxtDisplayTextLine(2,"Volumen = 2");
                                wait10Msec(10);
                                break;
                        case 3:
                                nxtDisplayTextLine(2,"Volumen = 3");
                                wait10Msec(10);
                                break;
                                case 4:
                                nxtDisplayTextLine(2,"Volumen = 4");
                                wait10Msec(10);
                                break;
                        default:
                                nxtDisplayTextLine(2,"Volumen = ALTO");
                                wait10Msec(10);
                        }// switch
                }// while
                nxtDisplayTextLine(2,"Volumen = BAJO");
        }// while 1
}// main
```

Unidad II: Lego NXT

«Brick del NXT, Botones del NXT, Altavoz, LCD, Actuadores y lineas de salida, Servomotores, LED's, Sensores, Interruptor, Micrófono, Infrarrojo Inactivo, Infrarrojo Activo, Sonar, Otros sensores.»

2 Introducción a la unidad

Desde la antigüedad, el ser humano ha soñado con crear seres inteligentes. Un ejemplo de ello se encuentra en la mitología griega: *Hephaestus,* hijo de *Hera*, creo un ser de bronce llamado *Talos*, cuya misión era proteger *Creta*. En términos modernos, podríamos afirmar que *Talos* es un androide de protección y servicio. Existen otros ejemplos como el Pato mecánico creado por Jacques de Vaucanson en el siglo XVIII.

Sin embargo, es hasta 1942 que surge el termino de **Robot**. Acuñado por Isaac Asimov en su famoso libro: "Yo robot". La palabra robot se desprende de la palabra en checo *robota* que significa "servidumbre" o "trabajo forzado".

Con el avance de la tecnología y el abaratamiento de los costos, la robótica ha dejado de ser una área exclusiva de las Universidades. Así, al finalizar esta unidad, el alumno podrá armar y programar sus propios robots usando como plataforma de desarrollo el lenguaje C y el kit comercial Lego NXT.

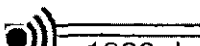
El Lego NXT es el último modelo de la serie de juguetes *MindStorms*, fabricada por la popular compañía LEGO. A pesar de ser un juguete, el NXT posee características muy particulares que lo hacen ideal para armar y programar pequeños robots. El NXT se puede conseguir a través de Internet o dentro del país en tiendas dedicadas, su precio oscila entre los $4000 y $5000 pesos (MX) pero varia según la paridad del dólar (US).

Para fines de su estudio, resulta conveniente agrupar los componentes del NXT de la siguiente manera:

1. Brick
2. Motores
3. Sensores
4. Piezas adicionales

2.1 El Brick del NXT

El Brick es el cerebro del NXT, por tanto es el componente más importante. En su interior se encuentran todos los componentes electrónicos necesarios para alimentar y controlar los motores y sensores que conformarán nuestro robot.



1988: Lego y el MIT colaboran en el desarrollo de un "ladrillo inteligente" que de vida a las creaciones de LEGO vía programación por ordenador. Así, en 2006 surge el NXT.

El Brick cuenta con dos procesadores: 1) Un AVR de 8 bits a 8Mhz destinado al control de los sensores y 2) un ARM7 de 32 bits a 48Mhz para la ejecución de nuestros programas y la comunicación con otros dispositivos.

Gracias a la memoria FLASH de 256 Kb que tiene integrada, el Brick puede almacenar una gran cantidad de programas. Lo que aunado a su memoria RAM de 256 Kb lo vuelve una excelente opción para programadores inexpertos pues no tendrán que preocuparse en ahorrar recursos de procesamiento.

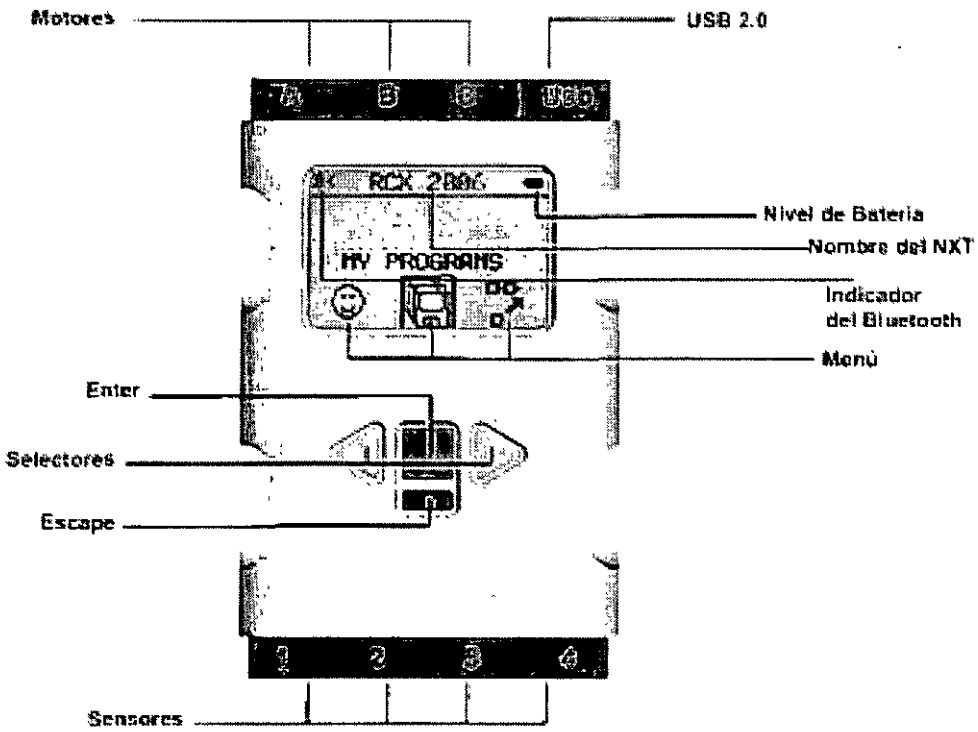
Para brindar una autonomía total al robot (fundamental en los robots móviles), el Brick cuenta con una batería recargable de 12 V a XXXXX amperes con la cual alimenta a todos los dispositivos conectados a él.

El NXT se puede comunicar con otros dispositivos (tales como computadoras, PDA, celulares) a través de un puerto USB 2.0 de alta velocidad o mediante tecnología Bluetooth Clase II con un alcance de hasta 10 metros.

Para facilitar la interacción con el usuario, el NXT tiene un Display LCD de 100*60 pixeles y un pad de 4 botones.

2.2 Botones del NXT

El NXT cuenta con una interfaz muy sencilla e intuitiva desde la cual el usuario puede configurar el brick, así como seleccionar el programa que desea ejecutar en el robot. El manejo de dicha interfaz se hace mediante cuatro botones que nombraremos de la siguiente manera: *Enter*, *Selectores* y *Escape*.



IMG 1: Partes del Brick

El usuario que cuenta con experiencia previa en el manejo de celulares, no encontrará ningún problema para interactuar con el NXT. Sin embargo, no esta de mas mencionar las principales instrucciones acerca del manejo del Brick.

A) Encendido / Apagado:
Para prender el Brick es necesario dejar presionado *enter* durante un par de segundos, hasta que se visualice en el Display una animación de bienvenida.
El apagado del brick se logra pulsando *escape* hasta que en el Display se nos pregunte si deseamos apagar el Brick, con los botones selectores elegimos OK y presionamos *enter.*

B) Correr un Programa
1)Encendemos el Brick. 2) Con los selectores elegimos el icono PROGRAMAS y presionamos *enter.* 3) Seleccionamos el programa que deseamos ejecutar y presionamos *enter,* 4) Seleccionamos la opción RUN y presionamos *enter.*

C) Detener un Programa
Durante la ejecución del programa, se presiona una vez *escape.*

D) Eliminar un Programa
Se repiten los 3 primeros pasos del punto B, se selecciona la opción *delete* y se presiona *enter.* Una vez eliminado un archivo, éste no podrá recuperarse.

E) Enviar un Programa
Se repiten los 3 primeros pasos del punto B, se selecciona la opción *send* , se elige el dispositivo al cual se quiere enviar el archivo y listo. Nota: Es necesario haber establecido la conexión previamente.

F) Activar / Desactivar Bluetooth
Seleccione la opción *settings* y elija la opción *bluetooth.* Seleccione *On* u *Off* según corresponda.

G) Establecer una conexión vía bluetooth
Ingrese al menú de opciones de Bluetooth y seleccione la opción *search,* espere a que el brick muestre todos los dispositivos disponibles. Seleccione el dispositivo al que desea conectarse y asigne una de las tres conexiones disponibles.

H) Restablecer el Brick
Si la batería se encuentra descargada, es posible que se congele la pantalla del brick, si esto sucede tan solo debe dejar presionado durante 5 segundos el botón *escape.*

La tecnología Bluetooth fue desarrollada por Sony con la finalidad de servir como estandar de comunicación para comunicar todos los aparatos del hogar y la oficina. El de esta tecnología proviene del rey danés del siglo X llamado Harold Bluetooth, conocido por unificar las tribus de Noruega, Suecia y Dinamarca.

2.2.2 Programación de los botones

El lenguaje RobotC nos provee de una librería que permite ver el estado actual de los botones del Brick, de forma tal que podemos determinar si el usuario ha presionado un botón y de esa forma realizar una función en especifico.

El manejo del PAD se realiza a través de la palabra reservada: **nNxtButtonPressed**, esta variable adquiere distintos valores según el botón que se encuentre presionado. El NXT sólo puede leer un botón a la vez, por lo cual no se admiten combinaciones.

Para distinguir un botón de otro, se recurre al *array* **TbuttonMasks**, cuyos estructura viene predefinida en el NXT y no es necesario escribirla en nuestros códigos.

```
typedef enum
{
  kNoButton      = 0xFF,
  kExitButton    = 0,
  kRightButton   = 1,
  kLeftButton    = 2,
  kEnterButton   = 3
} TbuttonMasks;
```

A continuación realizaremos un programa que cuenta el número de veces que ha sido presionado cada botón. Para ello haremos uso de una estructura *switch* en la cual compararemos el valor de nNXTButtonPressed.

```
// Contadores para cada boton.
int nLeftButton  = 0;
int nRightButton = 0;
int nEnterButton = 0;
int nExitButton  = 0;


task ButtonTask(){  // Esta función se ejecuta cuando se usa el PAD.

 //Aumentamos en uno el contador que corresponda al botón
presionado.
  switch (nNxtButtonPressed){
     case kLeftButton:           ++nLeftButton;
     break;
     case kRightButton:  ++nRightButton;          break;
     case kEnterButton:  ++nEnterButton;          break;
     case kExitButton:            ++nExitButton;
     break;
     }

     return;
}//task
```

```
task main(){
      //  Indicamos al NXT que queremos ejecutar automaticamente
          //  la función ButtonTask cada vez que se use el PAD
     nNxtButtonTask  = ButtonTask;

         //Indicamos cuantos clicks se deben hacer en escape para
        // abortar el programa. Por default el valor es 1
     nNxtExitClicks  = 3;

     while (true){
        nxtDisplayTextLine(2, "Izq: %3d",nLeftButton);
        nxtDisplayTextLine(3, "Der: %3d",nRightButton);
        nxtDisplayTextLine(4, "Ent: %3d",nEnterButton);
        nxtDisplayTextLine(5, "Esc: %3d",nExitButton);
        }
     return;
}//main
```

2.3 Altavoz del NXT

En el interior del Brick se encuentra un pequeño *Speaker* similar al que tienen las computadoras. A través de él se pueden emitir sonidos simples.

2.3.3 Programación del Altavoz

Se puede usar el altavoz del NXT para reproducir sonido durante nuestros programas. Para ello existen dos funciones, la primera nos permite emitir alguno de los sonidos pre configurados en el NXT. Con la segunda función podemos indicar la frecuencia y la duración de cada sonido.

La función **PlaySound()** reproduce un archivo de sonido compatible con le NXT. Recibe como parámetro alguna de las siguientes palabras:

soundBeepBeep	soundLast
soundBlip	soundLowBuzz
soundDonwardTones	soundShortBlip
soundException	soundUpwardTones

Crea un archivo nuevo en RobotC e implementa un programa que reproduzca tres de los sonidos predefinidos que quieras.

La función **PlayTone()** emite un sonido con frecuencia y duración definida a través de los parámetros que se le envían.

Esta función es de gran utilidad ya que se pueden sonidos distintos basados en el valor de una variable o de un sensor. A continuación mostraremos todo el potencial de la función reproduciendo una famosa obra de arte.

```
Task main(){
     wait10Msec(75);
     PlayTone(784,15);   wait10Msec(19);
     PlayTone(784,15);   wait10Msec(19);
     PlayTone(784,15);   wait10Msec(19);
     PlayTone(622,38);   wait10Msec(38);

     wait10Msec(38);
     PlayTone(698,15);   wait10Msec(19);
     PlayTone(698,15);   wait10Msec(19);
     PlayTone(698,15);   wait10Msec(19);
     PlayTone(587,38);   wait10Msec(38);

     wait10Msec(75);
     PlayTone(784,15);   wait10Msec(19);
     PlayTone(784,15);   wait10Msec(19);
     PlayTone(784,15);   wait10Msec(19);
     PlayTone(622,15);   wait10Msec(19);
     PlayTone(831,15);   wait10Msec(19);
     PlayTone(831,15);   wait10Msec(19);
     PlayTone(831,15);   wait10Msec(19);
     PlayTone(784,15);   wait10Msec(19);
     PlayTone(1244,15);  wait10Msec(19);
     PlayTone(1244,15);  wait10Msec(19);
     PlayTone(1244,15);  wait10Msec(19);
     PlayTone(1047,38);  wait10Msec(38);

     wait10Msec(19);
     PlayTone(784,15);   wait10Msec(19);
     PlayTone(784,15);   wait10Msec(19);
     PlayTone(784,15);   wait10Msec(19);
     PlayTone(587,15);   wait10Msec(19);
     PlayTone(831,15);   wait10Msec(19);
     PlayTone(831,15);   wait10Msec(19);
     PlayTone(831,15);   wait10Msec(19);
     PlayTone(784,15);   wait10Msec(19);
     PlayTone(1397,15);  wait10Msec(19);
     PlayTone(1397,15);  wait10Msec(19);
     PlayTone(1397,15);  wait10Msec(19);
     PlayTone(1175,38);  wait10Msec(38);

     wait10Msec(19);
     PlayTone(1568,15);  wait10Msec(19);
     PlayTone(1568,15);  wait10Msec(19);
     PlayTone(1397,15);  wait10Msec(19);
     PlayTone(1244,19);  wait10Msec(19);

     wait10Msec(19);
```

```
      PlayTone(1175,15);  wait10Msec(19);
      PlayTone(1568,15);  wait10Msec(19);
      PlayTone(1568,15);  wait10Msec(19);
      PlayTone(1397,15);  wait10Msec(19);
      PlayTone(1244,19);  wait10Msec(19);

      wait10Msec(19);
      PlayTone(1175,15);  wait10Msec(19);
      PlayTone(1568,15);  wait10Msec(19);
      PlayTone(1568,15);  wait10Msec(19);
      PlayTone(1397,15);  wait10Msec(19);
      PlayTone(1244,19);  wait10Msec(19);

      wait10Msec(38);
      PlayTone(1047,19);  wait10Msec(19);

      wait10Msec(38);
      PlayTone(784,38);   wait10Msec(38);
}//main
```

Problema 1

Escribe un programa que reproduzca un sonido distinto según el botón que el usuario presione en el PAD

Problema 2

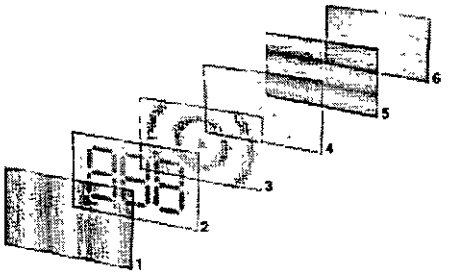
Escribe un programa que reproduzca la "Marcha Imperial" de la película StarWars

En la siguiente dirección muestran distintas formas de crear sonidos en el computador para despues ser reproducidos en el NXT: http://www.thenxtzoo.com/NXT_sound.html

2.4 LCD del NXT

Una pantalla de cristal liquido o LCD es una pantalla delgada y plana formada por un gran numero de pixeles colocados delante de una fuente de luz, mediante este mecanismo se puede dibujar un sin fin de formas. Su uso es muy extendido ya que son relativamente baratas y consumen muy poca energía.

En el caso concreto del NXT, el dsiplay tiene una extensión de 100*60 pixeles, lo que nos da una matriz constituida por 6000 puntos luminosos que podemos controlar a gusto para formar diferentes figuras.



IMG 2: Capas de un display LCD.

El NXT identifica cada pixel mediante sus coordenadas en la pantalla indicadas como X, Y. Es importante hace la aclaración de que el LCD del NXT tiene su origen en la esquina inferior izquierda, por lo que un pixel colocado en el centro de la pantalla tendría como coordenadas 50,30.

2.4.2 Programación del LCD

El compilador RobotC nos brinda un basto repertorio de funciones listas para ser utilizadas, desde como pintar un pixel hasta escribir texto en letras muy grandes. La primer función que todo programador debe saber es: **eraseDisplay()**, como su nombre lo indica, esta función borra todo lo que este dibujado en pantalla.

Lineas arriba, usamos la función **nxtDisplayTextLine()**, los parametros que recibe esta función son:

nxtDisplayTextLine(*linea*,"*texto a imprimir*", *NombreVariable*, *VariableDos*);

La pantalla se divide automáticamente en 7 renglones que se numeran del 0 al 6. Si se desea imprimir el valor de una variable se utiliza **%d** que indica al NXT que deberá sustituir esos dos caracteres por el valor de la variable indicado en el tercer o cuarto parámetro.

Otra forma de escribir texto, es indicar las coordenadas en las que queremos que se empiece a imprimir. Esto se logra mediante la función **nxtDisplayStringAt()**. Que se diferencia de la anterior en que al principio recibe dos parametros: X y Y.

Una alternativa a esta función es **nxtDisplayBigStringAt()** cuyo funcionamiento es igual pero escribe letras más grandes.

1962: Richard Williams de RCA encontró que había algunos cristales líquidos con interesantes características electro-ópticas y se dió cuenta del efecto electro-optico mediante la generación de patrones de bandas en una fina capa de material de cristal líquido por la aplicación de un voltaje. Este efecto se basa en una inestabilidad hidrodinámica formada, lo que ahora se denomina "domimnios Williams" en el interior del cristal líquido.

Las funciones de dibujo solo se encuentran disponibles a partir de la versión 1.5 del compilador RobotC. Estas incluyen círculos, cuadrados, rellenos, lineas y puntos. Sin embargo, resulta más práctico que sean abordadas en el apartado siguiente.

Problema 3

Escribe un programa que simule los créditos de una película Se deben mostrar al menos 10 nombres con sus respectivos cargos dentro de la producción.

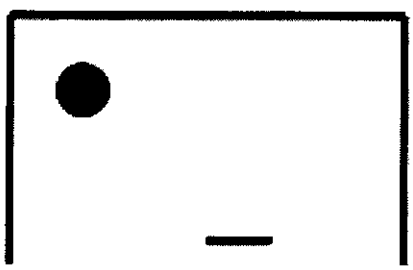
2.5 Aplicación Practica

Durante este apartado desarrollaremos desde cero una aplicaciones más grande y que requiere de un nivel superior de programación del que veníamos usando. Se trata de una versión simplificada del clásico juego de Ping Pong de los años 70. Para crearlo haremos uso de funciones que no habíamos mencionado así que pon atención para reconocerlas y aprender a usarlas.

2.5.1 Un sencillo juego de Ping Pong

Nuestro juego consistirá en una pelota que estará rebotando por todo el escenario, conformado por un rectángulo al cual se le ha cortado el arista inferior. Si la pelota toca el área que corresponde a ese arista, el jugador habrá perdido, para evitarlo, el usuario puede mover de forma horizontal una linea controlada mediante el PAD.

Nuestro escenario deberá ser similar al de la siguiente imagen:



IIMG 3: Pantalla de Juego

Lo primero que debemos hacer es determinar como controlaremos el movimiento de la pelota. ¿se te ocurre alguna idea?

Siempre es más fácil programar videojuegos si pensamos como si estuvieran sucediendo en la vida real, pensemos ¿Que hace que las pelotas caigan? Así pues, nos hace falta una variable que sirva como gravedad. Pero nuestra pelota debe ser capaz de rebotar, por lo que nos falta otra variable que sirva como fuerza.

Si nos fijamos en estas dos variables, nos daremos cuenta que corresponde al eje Y y al eje X respectivamente. Como nuestra pantalla es plana, no hace falta incluir un tercer eje, por el momento vamos bien.

```
byte dy=1;  // direccion del balon en y
byte dx=1;  // direccion del balon en x
```

Las variables dy y dx adquieren el valores comprendidos del -1 al 1. En el caso de dy, -1 significa un movimiento hacia abajo mientras que en dx -1 es un movimiento a la izquierda.

Como nuestro movimiento es dinámico a lo largo del programa, la posición de nuestra pelota no puede ser fija, debemos determinar dos variables que indicaran donde está nuestra pelota. El valor de estas variables se verá afectado por dy y dx.

```
byte px=30; // posicion de la pelota
byte py=50;
```

Al igual que la pelota, nuestra linea inferior se mueve, por tanto tambien requerimos dos variables para ella:

```
byte rx=20; // posicion de la regla
byte ry=6;
```

Una vez determinadas las variables que vamos usar, es momento de imprimir en pantalla las paredes que conforman nuestro escenario de juego. Para ello usaremos la función: nxtDrawLine(), Esta función recibe como parámetros la posición (X,Y) donde inicia la línea y la posición (X,Y) donde termina.

Para pintar las paredes usaremos las siguientes coordenadas:

```
nxtDrawLine(0, 0, 0, 64);      // Pared izq
nxtDrawLine(99, 0, 99, 64);  // Pared der
nxtDrawLine(0, 55, 99, 55);  // Pared sup
```

Para pintar la pelota en el escenario, usaremos dos funciones. La primera es **nxtDrawEllipse()**, esta función pinta un elipse basado en las dos posiciones que se indican como parámetros. La segunda función es: **nxtFillEllipse()**, rellena un elipse previamente dibujado. Para fines practicos, nuestra pelota tendra un diametro de 8 pixeles pero si asi lo deseas puedes cambiarla a tu gusto.

```
//--- pelota
 nxtDrawEllipse(px,py,px+8,py-8);
 nxtFillEllipse(px,py,px+8,py-8);
```

Es momento de hacer que la pelota se mueva por el escenario, para lo cual usaremos dos sentencias que harán que el valor de las variables px y py cambien según dx y dy.

```
//--- movimiento de la pelota
px=px+(dx);
py=py+(dy);
```

Si utilizamos paréntesis, obligamos al NXT a hacer una suma algebraica.

Hasta ahora nuestra pelota solo se mueve infinitamente en una dirección, es por eso que debemos determinar en que momento se estrella con alguna de las paredes para que cambiemos el valor de dx y dy de forma tal que se visualice el efecto de rebote.

Determinar que nuestra pelota choco es bastante facil, basta conque comparemos la posicion px y py con la X y la Y donde estan impresas las paredes:

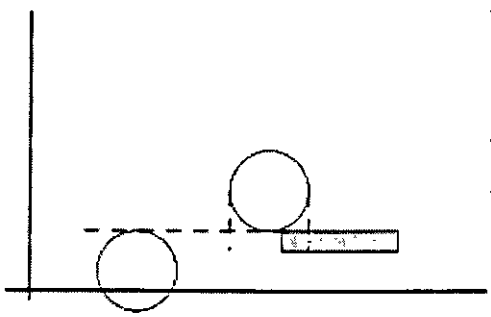
```
//--- Rebote
if(px<=1){ // muro izq
    dx=1;
    PlaySound(soundShortBlip);
}

if(px>=90){ // muro der
  dx=-1;
  PlaySound(soundShortBlip);
}

if(py>=54){ // techo
  dy=-1;
  PlaySound(soundShortBlip);
}
```

El siguiente paso es dibujar en pantalla la regla con la cual el usuario podrá evitar que la pelota salga del área de juego. Podríamos usar la función nxtDrawLine() para imprimir la regla, pero recomiendo usar **nxtDrawRect()** ya que nos permite dibujar una regla mas "solida". Esta función pinta un rectángulo que recibe como parámetros el punto de su esquina superior izquierda y el de su esquina inferior derecha.

```
nxtDrawRect(rx,ry,rx+la,ry-1); // regla
```



IMG 4: Calculo de colisión

Ahora es necesario saber si la pelota choco con la regla inferior. Su calculo es un poco complicado pues este se da sólo cuando el punto Y inferior de la pelota (y-8) coincide con la altura Y superior de la barra y al mismo tiempo el punto X de la pelota está a la altura del punto X de la regla sin que el punto X derecho de la pelota (x+8) supere el punto X derecho de la barra (rx+15).

Nuestro código para detectar una coalición con la regla quedaría así:

```
//contacto con regla
if((py-8)<=ry){
 if( (px>=rx) && (px+8<=rx+20) ){
      dy=dy*-1;
      PlaySound(soundShortBlip);
 }
}
```

Determinar cuando el usuario pierde el juego es muy fácil, pues basta conque la posición en Y de la pelota sea inferior a la posición en Y de la regla:

```
//-- PERDIO EL JUEGO
if(py<ry){
 eraseDisplay();
 nxtDisplayBigStringAt(10,40,"PERDIO!");
 PlaySound(soundDownwardTones);
 wait10Msec(300);
}
```

Como dijimos en un principio, la regla se moverá de conformidad a lo que el usuario le indique a través de los botones seleccionadores, para ello usaremos las siguientes lineas de código:

```
//-- control
 if(nNxtButtonPressed>0){
      if(nNxtButtonPressed==kLeftButton){
           if(rx>3){
              rx=rx-4;
                   }
      }
      if(nNxtButtonPressed==kRightButton){
           if(rx<(97-la)){
              rx=rx+5;
                    }
      }
    wait10Msec(2); //evitar que se interpreten muchos clicks seguidos
 }
```

Para finalizar nuestro programa solo falta añadir un trozo de código que se encargara de actualizar en pantalla el estado de nuestros componentes (pelota y regla).

```
//-- borro pantalla
 wait10Msec(250);  eraseDisplay();
```

Mientras menos tiempo se marque en la función wait10msec() más rápido y dificil será nuestro juego.

Al final, nuestro código debió haber quedado así:

```
task main(){
     byte rx=20; //poscicion de la regla
     byte ry=6;

     byte px=30; //posicion de la pelota
     byte py=50;

     byte dy=1; //direccion del balon en y
     byte dx=1;  //direccion del balon en x

  while(true){
    //-- control
     if(nNxtButtonPressed>0){
          if(nNxtButtonPressed==kLeftButton){
               if(rx>3){
                  rx=rx-4;
                       }
          }
          if(nNxtButtonPressed==kRightButton){
               if(rx<(97-la)){
                  rx=rx+5;
                        }
          }
        wait10Msec(2); //evitar que se interpreten muchos clicks seguidos
     }

  //-- pantalla estatica
      nxtDrawLine(0, 0, 0, 64);      // izq
      nxtDrawLine(99, 0, 99, 64); // der
      nxtDrawLine(0, 55, 99, 55); // sup

  //-- pantalla dinamica
      nxtDrawRect(rx,ry,rx+la,ry-1); // regla

  //--- pelota
      nxtDrawEllipse(px,py,px+8,py-8);
      nxtFillEllipse(px,py,px+8,py-8);

  //-- movimiento de la pelota
      px=px+(dx);
      py=py+(dy);

  //-- rebote en paredes y techo
      if(px<=1){ //muro izq
          dx=1;
          PlaySound(soundShortBlip);
      }
      if(px>=90){ //muro der
        dx=-1;
```

```
        PlaySound(soundShortBlip);
      }

      if(py>=54){ //techo
        dy=-1;
        PlaySound(soundShortBlip);
      }

      //contacto con regla
      if((py-8)<=ry){
       if( (px>=rx) && (px+8<=rx+20) ){
            dy=dy*-1;
            PlaySound(soundShortBlip);
       }
      }

      //PERDIO EL JUEGO
      if(py<ry){
       eraseDisplay();
       nxtDisplayBigStringAt(10,40,"PERDIO!");
       PlaySound(soundDownwardTones);
       wait10Msec(300);
      }

      //-- borro pantalla
    wait10Msec(250);  eraseDisplay();
 }//while
}//main
```

Problema 4

Al programa anterior, agrega una función que permita al usuario poner en pausa el juego mediante el botón enter del PAD. .

Problema 5

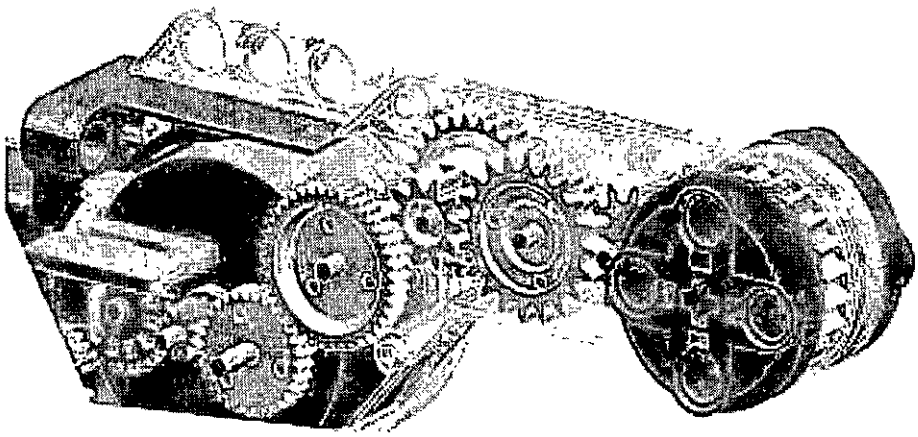
Mejora el programa anterior para que cada 15 golpes que de la pelota con la regla, el usuario pase a un nivel más difícil Cada nivel debe de ir más rápido la pelota.

2.6 Lineas de Salida del NXT

2.6.2 Servomotores

La diferencia entre un motor y un servomotor radica en que el motor tan sólo es una bobina que gira en un sentido u otro dependiendo de la corriente que se le aplique, mientras que un servomotor incluye un juego de engranajes y aun más importante, tiene un dispositivo electrónico de control.

La principal ventaja de los servomotores es que podemos detener el motor en un punto especifico del giro, e incluso sabemos en todo momento cuantas revoluciones ha dado desde que fue encendido y de esa forma calcular la distancia recorrida por el robot.



IMG 5: Interior de un Servomotor NXT

Los servomotores del NXT se conectan en los puertos A, B y C. A efectos de código, reciben el nombre del puerto según al cual están conectados, de esta forma, si quiero saber a cuanta velocidad esta girando el motor conectado al puerto C invocaría la función **motor();** pasando como parámetro "motorC": **motor(motorC);**

La velocidad de los servomotores del NXT se indica mediante valores que pueden ir del -100 al 100, por lo cual el 0 significa que el motor se encuentra inactivo y los extremos (100 y -100) son la máxima velocidad alcanzada en un sentido u el otro.

El giro del servomotor se controla mediante el conteo de ***encoders***. Un *encoder* equivale a un grado de giro, por lo cual 360° son 360 *encoders.* Es imprescindible saber que el conteo de *encoders* no es infinito, al llegar a +/- 32 768 éste regresa su valor a 0, por lo que el programador debe tomar sus precauciones al usar este sistema.

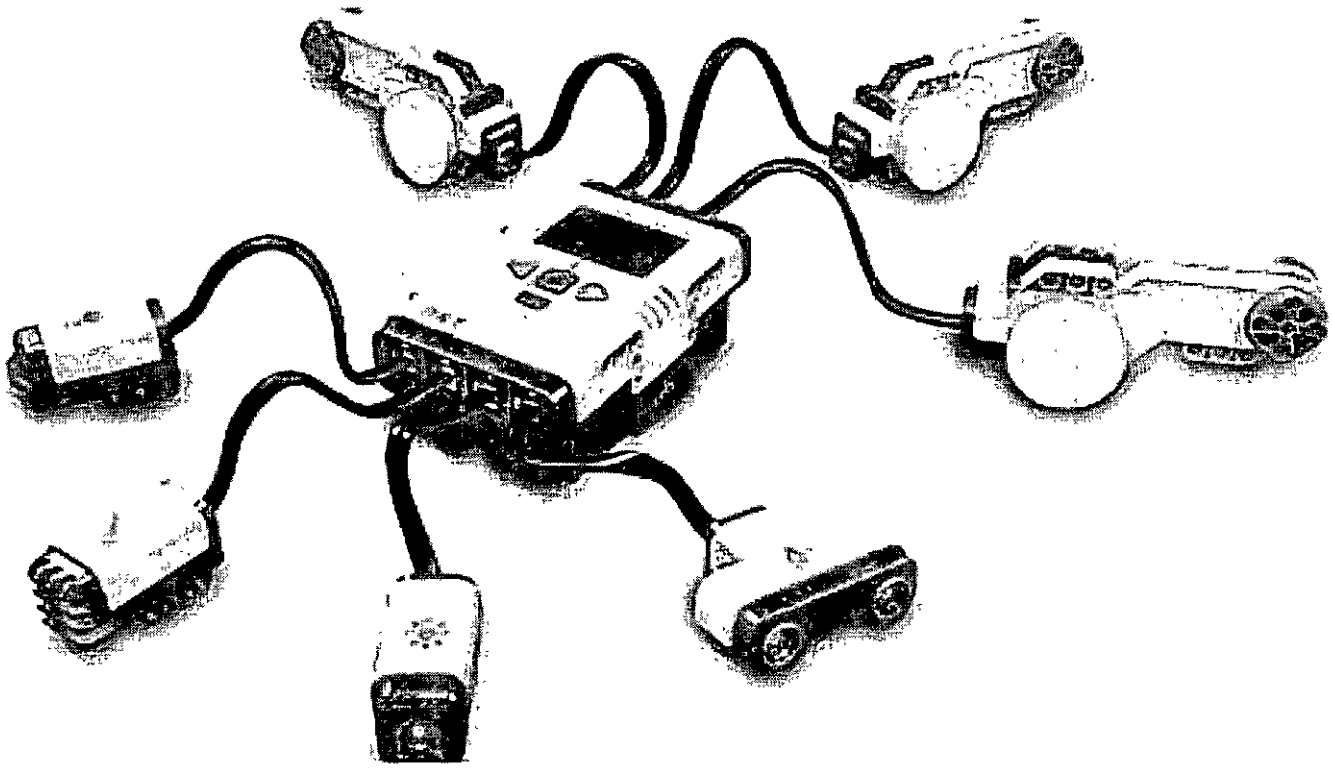
En el siguiente ejemplo, el motor conectado al puerto A, girara durante 50 encoders a una velocidad de 75%, luego, se detendrá y emitirá un sonido descendente.

```
task main(){
    while(nMotorEncoder[motorA]<50){
        motor[motorA]=75;
    }
   PlaySound (soundLowBuzz);
}
```

2.6.3 LED's

El NXT permite conectar diodos LED a sus lineas de salida. Se programan igual que los motores. Por ejemplo, para encender un LED conectado al puerto B podríamos usar el siguiente código:

```
task main(){
    for(;;){
        motor[motorB]=100;
    }
}
```



IMG 6: Partes Integrantes del NXT

2.7 Lineas de Entrada del NXT

El NXT cuenta con cuatro puertos (S1,S2,S3 y S4) que son usados como lineas de entrada, se diferencian de las lineas de salida en que estas tienen dos entradas de cable más, ya que no solo reciben energía sino que también envían información al Brick.

En estos puertos se conectan los diferentes tipos de sensores que son soportados por el NXT. En el kit estándar se incluyen un sensor Sonar, dos Touch, un IR y un Sound. Sin embargo, también se pueden encontrar en el mercado sensores como brújula, giroscopios, velocímetros, entre otros. Un sensor es un dispositivo diseñado para recibir información de una magnitud del exterior y transformarla en pulsos eléctricos que puedan ser cuantificados y manipulados.

Normalmente son construidos mediante la utilización de componentes pasivos que varían su conductividad eléctrica según estén sujetos a ciertas condiciones del entorno tales como la temperatura o la intensidad de luz. Algunos tipos de sensores son:

Fotoeléctricos: La construcción de este tipo de sensores, se encuentra basada en el empleo de una fuente de señal luminosa (lamparas, LED,etc) y una célula receptora de dicha señal, tales como pueden ser fotodiodos, fototransistores o LDR.

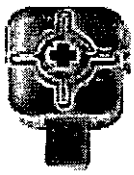
Sensores de contacto: Estos dispositivos son los más simples ya que simplemente se tratan de interruptores que se activan o desactiva si se encuentran en contacto con un objeto.

Sensores ultrasónicos: Este tipo de sensores se basan en el mismo funcionamiento que los del tipo fotoeléctrico, con la diferencia de que emiten una señal de tipo ultrasónica que es recibida por un receptor.

Para usar un sensor, es necesario declararlo antes de hacer alguna referencia a él. Para hacerlo basta usar la función: **SetSensorType(*port*,*type*);** El parámetro *port* hace referencia al puerto en el cual está conectado nuestro sensor, mientras que en *type* se envía una de las siguientes palabras reservadas, según corresponda al sensor que estemos usando:

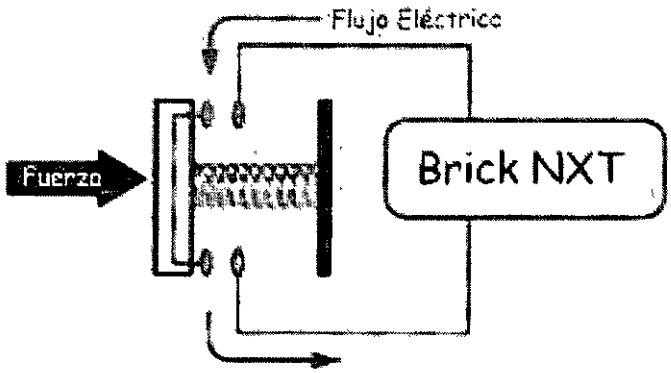
Tipo de Sensor	Palabra Reservada
Sensor Ultrasónico	sensorSONAR
Sensor Infrarrojo Pasivo	sensorLigthInactive
Sensor Infrarrojo Activo	sensorLigthActive
Sensor Interruptor	sensorTouch
Sensor de Sonido	SensorSound
Cualquier otro	sensorSONAR9V

2.7.2 Interruptor o Touch

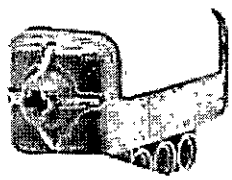


El sensor Touch o de tacto es el más simple que puede haber, al igual que todos los sensores debe ser declarado antes de usarlo y no hay restriccion alguna a la hora de conectarlo al Brick pues se puede conectar a cualquiera de sus lineas de entrada.

Los valores que devuelve son: 0 y 1 según su estado. Cuando no se encuentra presionado, se dice que su estado es *false*. Si se encuentra presionado su estado es *true*.



IMG 7: Circuito del Sensor Touch

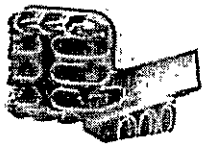


Este sensor consiste en un circuito muy simple; el flujo eléctrico es cortado en uno de sus puntos y se coloca un puente en la parte con la que hace contacto el sensor, cuando esta es presionada, se cierra el circuito y el valor del sensor pasa a verdadero, un resorte regresa al botón a su posición original.

En el siguiente ejemplo reproduciremos un sonido siempre que este presionado el sensor Touch.

```
task main(){
  SetSensorType(S3,sensorTouch);
    while(true){
     if(SensorValue[S3]==true){
       PlayImmediateTone(650,10);
     }
    }
}
```

2.7.3 Micrófono o Sound



El micrófono o sensor Sound nos permite, tal como lo indica su nombre, medir la cantidad de sonido que existe en el ambiente. Aunque teóricamente es posible usar el sensor para dar ordenes orales al robot, esto resulta muy complicado pues el sensor tan solo mide la cantidad de decibeles y no distingue entre un tono y otro.

Los valores que son devueltos por este sensor van del 0 al 100%, siendo el cero la ausencia de ruido en el ambiente. Se puede conectar a cualquier puerto y debe ser declarado antes de ser usado.

El siguiente ejemplo activa los motores conectados al robot siempre que exista una fuente de ruido superior al 50 %.

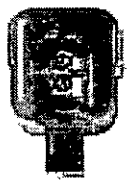
```
task main(){
  SetSensorType(S4,sensorSoundDB);
    while(true){
     if(SensorValue[S4]>50){
        nxtDisplayString(2, "%3d DB", SensorValue[S4]);
        motor[motorA]=75;
        motor[motorB]=75;
     } else{
        nxtDisplayString(2, "%3d DB", SensorValue[S4]);
        motor[motorA]=0;
        motor[motorB]=0;
     }
    }
}
```

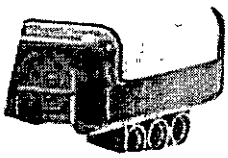
Problema 6

Escriba un programa que haga que el robot avance a una velocidad variante según la cantidad de ruido en el ambiente: a más ruido más velocidad, pero si encuentra un obstáculo debe de dar vuelta hacia un lado y continuar haciendo su función; los datos de los sensores deben ser impresos en pantalla.

2.7.4 Infrarrojo o IR

El sensor infrarrojo o *Infrared* (IR) cuantifica la cantidad de luz infrarroja en el ambiente. Dicha función la puede efectuar de dos formas: mediante recepción o mediante reflejo. La primera simplemente se limita a medir la luz ambiente, en la segunda enciende un diodo ubicado encima del receptor, la luz emitida rebota en los objetos cercanos y es captada por el receptor. Estos estados reciben el nombre de **Inactive** y **Active** según corresponde.





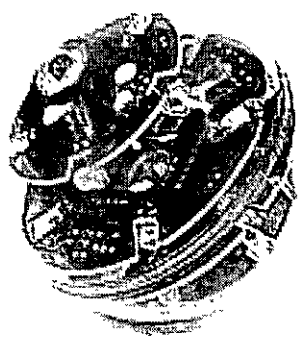
El sensor se puede conectar en cualquier puerto y devuelve valores comprendidos entre 0 y 100. En su estado Active el sensor puede distinguir (deficiente mente) los colores de una superficie plana si la luz es constante. Sin embargo, se recomienda adquirir el sensor de color (se vende por separado) si se desea realizar esta acción.

Es muy importante mencionar y tener en cuenta a la hora de programar nuestros robots que por ningún motivo se debe usar el sensor IR bajo luz solar directa al receptor, ya que este podría dañarse.

El siguiente ejemplo muestra en pantalla el nivel de luz captada por el sensor al tiempo que reproduce un sonido variable según el valor del sensor:

```
task main(){
  SetSensorType(S2,sensorLigthInactive);
    while(true){
     nxtDisplayString(2, "IR: %3d", SensorValue[S2]);
     PlayImmediateTone(SensorValue[S2],10);
    }
}
```

El sensor IR es muy socorrido en el juego de fútbol, debido esto a que se utiliza en este tipo de competencias una pelota de emisión infrarroja que emite esta luz invisible en todas direcciones, así los robot pueden ir hacia ella y tratar de llevarla a la portería.



IMG 8: Pelota Infrarroja oficial del Robocup

La longitud de onda de los rayos infrarrojos es menor que la de las ondas de radio y mayor que la de la luz visible. Oscila entre 10 a la -6 y 10 a la -3 centimetros.

Ahora vamos a realizar un programa que siga una luz infrarroja. Para ello es necesario tener conectado un motor en el puerto A y otro en el puerto B, así como el IR en el puerto S1 y configurado en su modo Active.

```
task main(){
  SetSensorType(S2,sensorLigthActive);
    while(true){
     if(SensorValue[S2]>70){
        nxtDisplayString(2, "IR:%3d ENC", SensorValue[S2]);
        motor[motorA]=0;
        motor[motorB]=0;
     }else{
        nxtDisplayString(2, "IR:%3d BUS", SensorValue[S2]);
        motor[motorA]=60;
        motor[motorB]=-60;
     }
    }
}
```

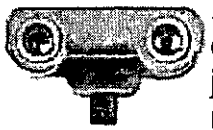
La gran limitación del sensor infrarrojo consiste en que sólo devuelve un valor que indica la presencia de luz y en que cantidad, pero no nos dice en que dirección se encuentra la fuente luminosa. Actualmente existe en el mercado un sensor que nos devuelve la dirección desde la cual proviene la luz expresada en un rango de 0 a 150 grados. Este sensor es desarrollado por MindStorms y recibe el nombre de IR Seeker.

Problema 7

Realiza un programa que haga que el robot trate de seguir tu mano cuando esta se encuentre cerca de su sonar, y que cuando se encuentre muy cerca de ella se detenga y espere, si la mano se aleja demasiado, debe de buscarla. Nota: Debido a las rugosidades de tu mano, es posible que no funcione del todo bien. Te recomiendo usar un cuaderno que simule ser tu mano.

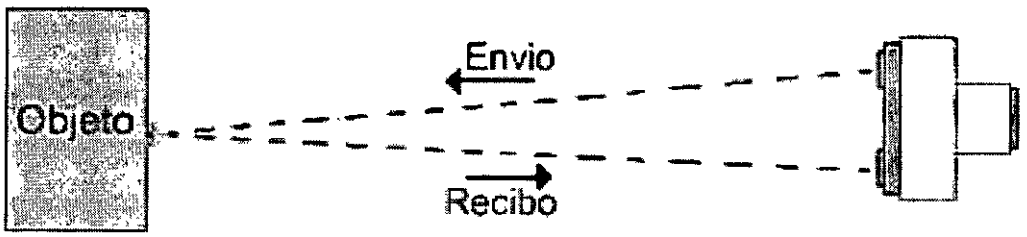
Dispositivos infrarrojos como los empleados durante la II Guerra Mundial permiten ver objetos en la oscuridad. Estos instrumentos consisten básicamente en una lampara que emite un haz de rayos infrarrojos, a veces denominados luz negra, y un telescopio que recibe la radiación reflejada por el objeto lo convierte en una imagen visible en un monitor.

2.7.5 Ultrasónico o Sonar



El sensor sonar se puede identificar por su forma rectangular que tiene dos círculos que en ocasiones dan la apariencia de ser una cabeza de juguete. El funcionamiento de este sensor se basa en la emisión y la recepción de ultrasonido.

El sensor emite una frecuencia continua de sonido ultrasónico que al rebotar con los objetos cercanos regresa al sonar, que de manera automática calcula el tiempo que tardó la señal en ir y regresar, con lo cual determina la distancia a la que se encuentra el objeto.



IMG 9: Funcionamiento del sensor Sonar

El máximo alcance del sonar es de 2.5 metros lineales y hacia el frente del sensor sonar. Cuando no existe ningún objeto con el cual rebote la frecuencia ultrasónica, el sensor devuelve un *indeterminado* representado por el numero 255.

Veamos un uso practico del sensor:

```
task main(){
  SetSensorType(S1,sensorSONAR);
    while(true){
     nxtDisplayString(1, "%3d", SensorValue[S1]);
     PlayImmediateTone(SensorValue[S1],10);
    }
}
```

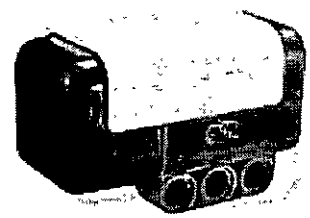
Combinado con otros sensores y con imaginación y creatividad, es posible realizar muchas cosas con el sensor Sonar, por ejemplo: un radar. Para realizarlo solo necesitas una base que pueda girar en su propio eje, colocar el sensor sonar en ella e ir tomando mediciones cada 4 o 5 encoders e irlas graficando en el Display del NXT

El sonido se desplaza a una velocidad de 334 metros por segundo en el aire a una temperatura de 20°C, lo que significa que la tasa de medicion del sensor Sonar es de tan solo 0.0145 segundos, que es el tiempo que le toma al sonido ir y venir en una distancia de 2.5 metros. (en total 5 metros.)

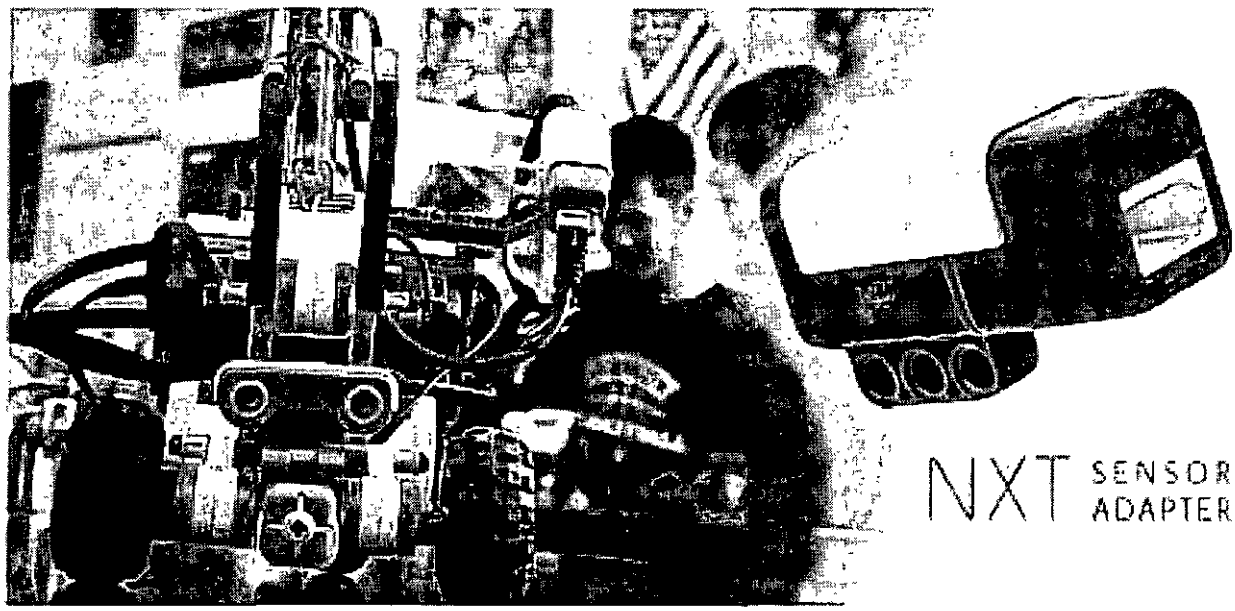
2.7.6 Otros Sensores

Tanto MindStorms como otras compañías han producido una gran variedad de sensores compatibles con el NXT. Algunos de ellos son:

- Giroscopio
- Velocímetro
- Brújula
- Color
- IR Seeker
- Termómetro



Debido a que el objeto de este curso son las piezas incluidas en el kit estándar del NXT, no abordaremos el uso de estos sensores. Bastará decir que los tres primeros se conectan al puerto S1 del Brick, mientras que los otros no tienen restricción.



También existe en el mercado un adaptador de sensores, con el cual se pueden conectar sensores de otros robots. Soporta hasta treinta dispositivos diferentes y su precio es de aproximadamente $39 dolares (US).