Proyectos de Robótica Con el Mindstroms NXT de LEGO (Nivel Básico)

Dr. Daniel Zaldivar Navarro
Dr. Erik V. Cuevas Jimenez
Dr. Marco Antonio Pérez Cisneros

Grupo de Robótica y Sistemas Inteligentes
Centro Universitario de
Ciencias Exactas e Ingenierias
Universidad de Guadalajara.

Prefacio

Durante el paso de los años se ha vinculado como motor de crecimiento de los principales países desarrollados, el uso de nuevas tecnologías las cuales promueven el desarrollo tecnológico, industrial y por ende económico de una sociedad.

Se deduce entonces que las sociedades mas modernizadas y con un mayor nivel de vida económica son también las más tecnificadas, es decir las que poseen un mayor numero de industrias dedicadas a la producción de aparatos, maquinaria que por sus características especiales permiten la solución a problemas clásicos mediante soluciones altamente tecnificadas.

En este sentido, en el contexto internacional es indiscutible el aprendizaje y aplicación de tecnologías como la robótica para la obtención de una sociedad altamente tecnificada.

Una estrategia muy exitosa utilizada en países altamente desarrollados y tecnificados (Japón, Alemania, etc.) con el objetivo de que una sociedad alcance un desarrollo importante y contribuya a la tecnificación, es la impartición temprana de tecnologías como la robótica y sus principios, presentada de una manera practica, amigable y haciendo uso de los conocimientos teóricos impartidos en materias de las ciencias exactas (matemáticas, física, computación, etc.).

En este contexto, se propone el presente material enfocado al aprendizaje de los principios de la robótica, mediante practicas amigables que utilizan y reafirman los conocimientos adquiridos en cursos de programación en alto nivel desde el sistema de educación de nivel medio superior hasta nivel ingeniería, licenciaturas en computación y a carreras afines. Así se propone la implementación de una serie de proyectos de robótica utilizando el sistema LEGO NXT de MINDSTROMS.

Conociendo el sistema

Mindstorms NXT de LEGO

Parte 1

Instalación del Software y Hardware

1. Partes del sistema LEGO Mindstorms NXT

A continuación se hace una presentación de los componentes principales del Mindstroms:

NXT (Cerebro electrónico)

Actuadores (Motores)

Sensores

Accesorios

2. Instalación software de LEGO Minstorms NXT

La instalacion del software de LEGO Mindstorms NXT se divide en 2 partes, la instalacion del controlador y el *Bricx Command Center*.

2.1 Instalación del controlador del NXT

El controlador para el NXT se puede descargar desde el CD de instalacion o de la siguiente liga:

http://mindstorms.lego.com/support/updates/

Lleva por nombre MINDSTORMS NXT Driver v1.02. Se debe hacer clic en la descarga para PC y elegir guardar en el disco duro. Se extrae el archivo, se ejecuta Setup.exe y se siguen las instrucciones.

2.2 Instalando Bricx Command Center

Bricx Command Center es el software que provee de una interfaz y de los programas necesarios para programar al NXT. Este se puede descargar de la siguiente dirección:

http://bricxcc.sourceforge.net/

En este material se usará la versión 3.3 *Bricx Command Center*

2.2.1 Configurando el Bricx Command Center

Al ejecutar Bricx, lo primero que observamos es una ventana de diálogo (figura 1) la cual configuraremos de la siguiente forma:

Port: USB
Brick Type: NXT
Firmware: Standard

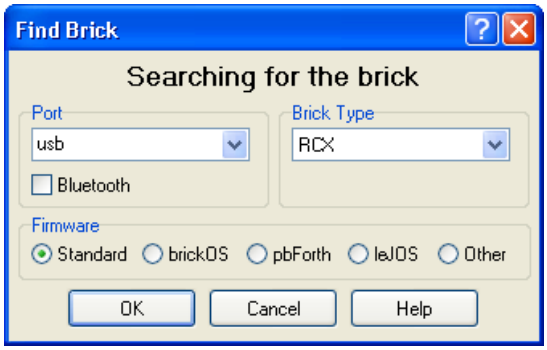


Figura 1. Ventana de configuración del Bricx

Antes de dar clik en "ok" para aceptar esta configuración, debemos primero encender al NXT y conectarle un extremo del cable USB y el otro extremo al puerto USB de la PC. como lo muestra la figura 2.

La conexión del puerto USB del NXT y el USB de la PC solo se nesecitara durante la programación del NXT

Figura 2. Conexión del NXT a la PC

2.3 Instalando el NXC

El programa y los manuales del NXC se puede bajar desde la siguiente direccion:

http://bricxcc.sourceforge.net/nbc/

Programando el

Mindstorms NXT de LEGO

Parte 1

Primer proyecto

“Robot de avance”

1. Proyecto "Robot de avance"

Mediante este primer proyecto se pretende familiarizar al lector con el entorno de programación *Bricx Command Center* y el NXC (Not eXactly C), el cual se utilizara para programar todos los siguientes proyectos descritos en este material.

Por esta razón este primer proyecto tendrá como objetivo que el robot ejecute la sencilla tarea de desplazarse hacia delante durante tres segundos y luego se detenga.

Para este primer proyecto se ha escogido la configuración de robot mostrado en la figura 4. Cuya construcción se describe detalladamente en el apéndice A. Se recomienda seguir esta configuración puesto que se volverá a utilizar en futuros proyectos. La figura 5 muestra el robot real en base al NXT.

1. 1 Reglas de comportamiento del robot

Las reglas del comportamiento del robot puede ser descrito como sigue:

Primero **enciende** los motores, **espera** 3 segundos y luego **apaga** los motores.

1.2 Programa del robot

El programa para que el robot ejecute el comportamiento descrito en 1.1 se expone a continuacion:

```
#include "NXCDefs.h"

task main(){
  OnFwd( OUT_BC, 75 );
  Wait( 3000 );
  Off( OUT_BC );
}
```

Es muy importante que seleccione el tipo de archivo .nxc, ya que de lo contrario no se compilará correctamente.

No es necesario en este momento entender el código perfectamente, por ahora lo importante es que se pueda cargar al robot. Para esto, una vez que se ha establecido la conexión entre la PC y el NXT (mediante3 el cable USB), dentro de *Bricx* se debe abrir un nuevo archivo (Ctrl+N) y después escribir el código mostrado anteriormente y guárdelo como *tresseg.nxc* (Figura 3).

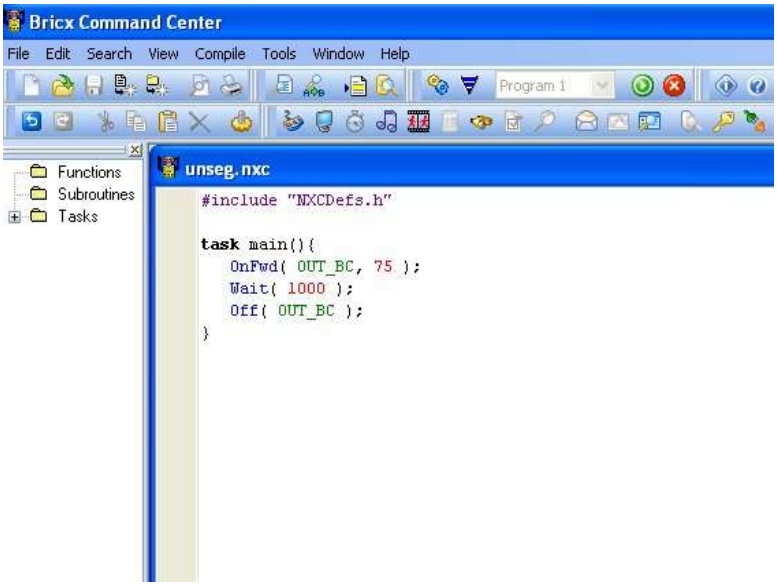


Figura 3. Primer programa en el NXT

Para compilar el programa presione el botón *Compile Program (F5)*, la compilación no debe de producir errores. Ahora que tenemos compilado el programa, es posible cargarlo al NXT. Basta con presionar *Download Program (F6)*, para que comience a descargarlo.

Ahora, en el menú del NXT seleccione *My Files*, esta opción contiene los archivos que se han cargado a la memoria del NXT. Seleccione *Software Files* y luego *tresseg* (el cual es el nombre que elegimos para el programa que creamosanteriormente). Finalmente *Run*, para correr el programa.

El robot debera ejectuar el objetivo planteado: avanzar durante tres segundos y luego detenerse.

Figura 4. Robot propuesto en base al sistema NXT de LEGO.

Figura 5. Robot NXT utilizado para el programa "tres segundos"

Parte 2

Segundo proyecto

“Robot explorador”

2. Proyecto " Robot explorador"

El robot explorador será un proyecto relativamente sencillo que correrá en el NXT y tendra como objetivo explorar el lugar en donde se encuentre, para esto el robot primero revisara si existen objetos u obstáculos delante de el, si no los hay avanzara, pero si los hay realizara un giro hacia la izquierda durante el tiempo que el obstáculo este presente.

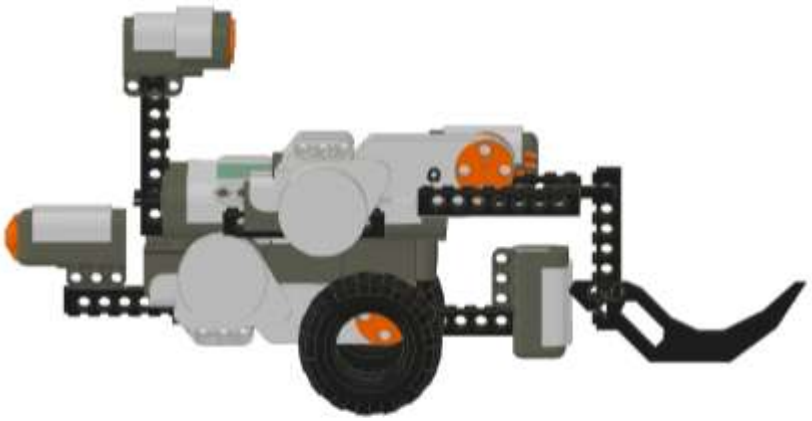
2. 1 Reglas de comportamiento del robot

Las reglas de comportamiento del robot seran las siguientes:

- Si **no hay obstaculo enfrente**, entonces **avanza**.
- De lo contrario, **gira a la izquierda.**

a) Explorador (vista trasera)

b) Explorador (Vista Frontal)



c)Explorador (vista Lateral)

d) Explorador (Isometrica)

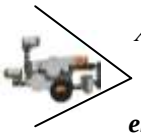
Como puede observarse se utiliza la misma configuración de robot utilizada en el primer programa y descrita en el apendice A. Las figuras 2.1 y 2.2 muestran el robot real.

2.2 Funciones NXC a utilizar

Para este proyecto, se necesitarán conocer las siguientes funciones del lenguaje NXC antes de programar el robot:

OnFwd(motores, fuerza)

Esta función enciende motores hacia delante, con fuerza de potencia. Los motores se mantendrán encendidos hasta que se les indique algo distinto. La potencia va de 1-100. Si los motores ya estaban encendidos, se actualiza la potencia. La nomenclatura para los motores se verá enseguida.



Al utilizar 'OnFwd" los motores se mantendran encendidos hasta que otra cosa se indique.

OnRev(motores, fuerza)

Realiza lo mismo que OnFwd, pero en sentido contrario.

Off(motores)

Apaga motores. Los mantiene atrancados en ese estado.

SetSensorLowspeed(puerto)

Esta función indicará al programa qué tipo de sensor contiene puerto.

SensorUS(puerto)

Devuelve el valor del Sensor Ultrasónico. Por default lo devuelve en centímetros (cm).

Wait(t)

Detiene la ejecución del programa con una pausa de t milisegundos.

Para los motores existen unas constantes. Estas comienzan por OUT_ seguidas de los motores a los que se requiera. Ejemplos:

Constantes:

OUT_A	Motor A
OUT_B	Motor B
OUT_C	Motor C
OUT_BC	Motores B y C
OUT_ABC	Todoslos motores

Para los sensores, la sintaxis es muy parecida:

IN_1	Sensor conectado a la terminal 1
IN_2	Sensor conectado a la terminal 2
IN_3	Sensor conectado a la terminal 3
IN_4	Sensor conectado a la terminal 4

2.3 Programa del robot

A continuación se muestra la codificación del programa utilizado para este proyecto. En el prgrama se consideró que el robot tiene conectado el motor izquierdo a la salida B y el derecho a la salida C, así como el sensor ultrasónico a la entrada 4. A continuación se presenta el codigo necesario para este programa.

```
#include "NXCDefs.h"

task main(){
	SetSensorLowspeed( IN_4 );

	while( true ){
		if( SensorUS( IN_4 ) > 40 )
			OnFwd( OUT_BC, 75 );
		else
			OnRev( OUT_B, 75 );
	}
}
```

Ahora es posible ejecutar el programa dentro del NXT, el cual hara que el robot sea capaz de explorar un lugar sin chocar.

2.3.1 Explicacion del codigo

Ahora se explicará el codigo utilizado en el programa para este proyecto:

```
#include "NXCDefs.h"
```

Esta es la cabecera del programa. Sirve para indicar que usaremos funciones para el NXT. Los programas siempre deben de contener esta primer línea.

```
task main(){
    …
}
```

Proceso principal. Desde aquí inicia la ejecución del todo el programa. Dentro de el se escriben todo el codigo que se desee ejecutar.

```
SetSensorLowspeed( IN_4 );
```

Inicializa el sensor ultrasónico en la entrada 4 del NXT.

```
while( true ){
    …
}
```

La estructura de control "while", sirve para ejecutar un bloque repetidamente mientras se cumpla la condición que está entre paréntesis.

```
if( SensorUS( IN_4 ) > 40 )
    …
else
    …
```

La instrucción "while" repite un bloque de instrucciones, hasta que desaparezca la condicion de disparo.

La instrucción dentro del **if**, se ejecuta siempre y cuando se cumpla la condición entre paréntesis. En este caso se trata de que el sensor ultrasónico detecte una cantidad mayor a 40cm.

La instrucción dentro del **else** se ejecuta si fue falsa la condición del **if**.

```
OnFwd( OUT_BC, 75 );
```

Enciende los motores B y C con una potencia del 75%

```
OnRev( OUT_B, 75);
```

Enciende el motor B en reversa con una potencia del 75%

2.3.2 Modificaciones al programa original

Sin embargo, es importante hacer notar que puede presentarse la ocasión de que el NXT quede muy cerca de obstáculos los cuales le imposibiliten girar. El robot en estos casos, quedaría atorado. Para evitar esta situación, se pueden hacer unas pequeñas modificaciones al programa original:

```
#include "NXCDefs.h"

task main(){
        SetSensorLowspeed( IN_4 );

        while( true ){
                if( SensorUS( IN_4 ) > 40 ) {
                        OnFwd( OUT_BC, 75 );
                } else {
                        OnRev( OUT_BC, 75 );
                        Wait( 200 );
                        while( SensorUS( IN_4 ) < 40 )
                                OnRev( OUT_B, 75 );
                }
        }
}
```

Lo que hacen estas nuevas líneas (indicadas en color amarillo), es echar en reversa al robot durante un pequeño momento (200 mili segundos) cuando este detecte algún obstáculo enfrente de el y luego hacer el giro correspondiente mientras el obstáculo siga presente.

En esta modificacion al codigo original se agregaron llaves {} al **if** y al **else**, esto es necesario debido a que ahora se tiene mas de una instrucción dentro del **else**.

Es necesario agregar corchetes ({}) a una instrucción if..else cuando se ejecuten mas de una instrucción en ellos.

Como ésta, puede haber muchas otras modificaciones que hagan que el robot explorador realice su tarea más eficientemente. Se recomienda que el lector experimente con otras modificaciones conservando el programa original y renombrando las modificaciones de prueba.

Figura 2.1: Robot explorador

Figura 2.1: Robot explorador

Parte 3

Tercer proyecto

“Robot sigue lineas”

3. Proyecto "Robot sigue líneas"

Este robot debe seguir una línea color oscuro (negro) la cual describe una trayectoria circular. La línea estará marcada sobre un plano de color claro (blanco). Dado que el NXT solo necesitará girar hacia un lado ya que la trayectoria es circular, las reglas del comportamiento se simplifican bastante como se muestra a continuación.

Finalmente la construcción detallada de este robot se expone en el apéndice B.

Figura 3.1: Robot sigue lineas

3.1 Reglas de comportamiento del robot

Las reglas del comportamiento del robot se expresan de la siguiente manera:

- **Mientras** el robot permanezca en la trayectoria, **avanzar**.
- **Si** el robot se sale de la línea, **girar a la izquierda** para corregir la trayectoria.

Se eligió la izquierda arbitrariamente, el lector puede hacerlo girar hacia la derecha si así lo desea.

3.2 Programa del robot

El codigo del programa para este proyecto es el siguiente:

```
#include “NXCDefs.h”
#define PW 50
#define THR 50

void avanza( ){
  OnFwd( OUT_BC, PW );
}
void girar_izq( ){
  OnFwd( OUT_C, PW );
  OnRev( OUT_B, PW );
}

task main ( ){
  SetSensorLight( IN_1 );

  while( true ){
    if( Sensor( IN_1 ) > THR )
      girar_izq( );
    else
      avanza( );
  }
}
```

3.3 Explicación del codigo:

```
#define PW 50
#define THR 50
```

Éstas dos líneas definen dos constantes. Asi cuando el codigo se ejecute y el compilador encuentre en cualquier parte del programa,

la constante PW, utilizara el valor con el cual se cargo al principio del programa, en este caso por un valor de 50. Lo mismo hará con THR.

El hacer uso de constantes definidas por el usuario facilita la depuración del programa.

Para esta aplicación PW se utiliza para definir la fuerza de los motores, asi por ejemplo, se podría necesitar cambiar la fuerza con la que se impulsarán estos motores. Ya no será necesario buscar y cambiar en cada línea que mueve a los motores, ahora sera suficiente con simplemente cambiar el valor de la constante al principio del programa y compilarlo de nuevo.

```
void avanza( ){
  OnFwd( OUT_BC, PW );
}
void girar_izq( ){
  OnFwd( OUT_C, PW );
  OnRev( OUT_B, PW );
}
```

El uso de funciones simplifica el programa, haciendo mas facil su comprensión y la busqueda de errores.

Aquí se definen dos funciones “avanza” y “ girar_izq”, las cuales ayudarán a simplificar la codificación del programa. Estas líneas se ejecutan solamente cuando son llamadas.

La primera funcion, “avanza” provoca que las salidas B y C se enciendan girando hacia delante. Mientras que “girar_izq” provoca que la salida C se encienda girando hacia delante y B girando hacia atrás.

```
if( Sensor( IN_1 ) > THR )
      girar_izq( );
   else
      avanza( );
```

Esta parte del código lee el valor del sensor infrarrojo “Sensor(IN_1)” si este es mayor que el umbral definido por THR la rutina “girar_izq” es activada, sino la rutina “avanza” es activada

A pesar de que el proyecto “sigue líneas” ejecuta adecuadamente las reglas de comportamiento descritas en 3.3 las reglas no contemplan que el robot se salga de su ruta por el lado izquierdo. Si esto pasara, el comportamiento que debería de realizar el robot es que regresara a su trayectoria girando hacia la derecha.

Sin embargo, debido a que el hardware de este proyecto solo tiene un sensor infrarrojo no es posible identificar en qué dirección debe girar el robot para corregir la trayectoria.

La solución es agregar un sensor infrarrojo además del que se tenia, y agregar el código que efectué los giros que correspondan según sea el caso, esta mejora se expondrá de manera detallada en el siguiente proyecto.

Parte 4

Cuarto proyecto

“Robot sigue líneas mejorado”

4. Proyecto "Robot sigue líneas mejorado"

Como se había ya mencionado al final de la sección 3.3 el proyecto "Robot sigue líneas" solamente funciona en una trayectoria que implique giros hacia un solo lado. Esto es debido a que se cuenta solamente un sensor, con este hardware el NXT es incapaz de conocer con certeza a dónde debería de girar para corregir la trayectoria.

Pero al incluir un nuevo sensor, la correccion en la trayectoria del robot sigue lineas es perfecta para cualquier caso.

4.1 Reglas de comportamiento del robot

Las reglas de comportamiento se expresan de la siguiente manera:

- ***SI*** *me encuentro dentro de la línea,* ***avanzar***
- ***SI NO***
  - ***SI*** *el último* ***sensor*** *en detectar fue el* ***izquierdo****,* ***girar*** *a la* ***izquierda***
  - ***SI NO girar*** *a la* ***derecha***

4.2 Programa del robot

El programa de este proyecto es el siguiente:

```
#include "NXCDefs.h"
#define BOUND 50
#define PWD 50

int ultimo = 0;
int dentro = 0;

void gira_izq(){
  OnFwd( OUT_C, PWD );
  OnRev( OUT_B, PWD - 25 );
}
void gira_der(){
  OnFwd( OUT_B, PWD );
  OnRev( OUT_C, PWD - 25 );
}
void avanza( ){
  OnFwd( OUT_BC, PWD );
}
```

```
task sensores(){
    SetSensorLight( IN_1 );
    SetSensorLight( IN_2 );

    while( 1 ){
      if( Sensor( IN_1 ) < BOUND )
        ultimo = 0;
      else if( Sensor( IN_2 ) < BOUND )
        ultimo = 1;

      if( Sensor( IN_1 ) > BOUND && Sensor( IN_2 ) > BOUND )
        dentro = 0;
      else
        dentro = 1;
    }
}

task main(){
  start sensores;

  while( 1 ){
    if( dentro == 0 ){
      if( ultimo == 0 )
        gira_izq();
      else
        gira_der();
    } else
      avanza();
  }
}
```

4.3 Explicacion del codigo

```
int ultimo = 0;
int dentro = 0;
```

Las variables declaradas fuera de las funciones, son validas en cualquier parte del programa

Éstas variables se encuantran declaradas fuera de cualquier función, por lo tanto tienen alcance en cualquier parte del programa.

La variable “ultimo”, nos servirá para identificar cuál fue el último sensor que detectó la línea. Esto le indicara al programa si el robot se salio de la linea por la izquierda o a la derecha. Para este proyecto y para utilizar una convencion, un valor de “0”, significará que el último sensor en detectar la línea fue el izquierdo, y el valor de “1” indicará que fue el derecho.

La variable “dentro” será la bandera que indique cuando el NXT se ha salido de la línea. Un valor de “1”, indica que el NXT aún detecta la línea, y un valor de “0”, que se ha salido de la linea.

```
void gira_izq(){
  OnFwd( OUT_C, PWD );
  OnRev( OUT_B, PWD - 25 );
}
void gira_der(){
  OnFwd( OUT_B, PWD );
  OnRev( OUT_C, PWD - 25 );
}
void avanza( ){
  OnFwd( OUT_BC, PWD );
}
```

En esta parte del codigo se declaran las funciones a utilizar para girar a la izquierda, derecha y avanzar, respectivamente.

```
task sensores(){
    SetSensorLight( IN_1 );
    SetSensorLight( IN_2 );

    while( 1 ){
      if( Sensor( IN_1 ) < BOUND )
        ultimo = 0;
      else if( Sensor( IN_2 ) < BOUND )
        ultimo = 1;

      if( Sensor( IN_1 ) > BOUND && Sensor( IN_2 ) > BOUND )
        dentro = 0;
      else
        dentro = 1;
    }
}
```

Lasd tareas o “task” se ejecutan siempre en paralelo junto con el “main”

Las actividades (**task**), son bloques que se ejecutan en paralelo, es decir, al mismo tiempo. El proceso principal (**main**), también es un task. En esta parte se define el task “sensores”, el cual estará constantemente actualizando los valores de “ultimo” y “dentro”.
Esta actividad pudo haberse declarado dentro de la misma actividad main, pero resulta ventajoso utilizarlo como una actividad independiente.

En este punto se puede vislumbrar el motivo de que se hayan declarado variables globales. Mientras que el task "sensores" se dedicará a actualizar los estados de "ultimo" (si es el sensor izquierdo o derecho) y "dentro" (si se esta o no sobre la linea), la actividad "main" decidirá qué instrucciones ejecutar según los valores de estos estados.

```
task main(){
  start sensores;

  while( 1 ){
    if( dentro == 0 ){
      if( ultimo == 0 )
        gira_izq();
      else
        gira_der();
    } else
      avanza();
  }
}
```

Es necesario iniciar las tareas o "tas" desde el main o alguna otra funcion..

La primer línea de esta parte del codigo "start sensores;" inicia la tarea definida para task "sensores". Es necesario iniciar los task desde main o alguna otra función.

Ahora, el objetivo de las estructuras if para esta parte del codigo, es si me encuentro fuera de la linea y el ultimo sensor en detectar fue el izquierdo, entonces la accion para corregir y volver a la trayectoria correcta es girar hacia la izquierda. Pero si el ultimo sensor en detectar fue el derecho, entonces girar a la derecha.
Si no hay cambios quiere decir que se esta sobre la linea entonce se debe seguir avanzando por esa trayectoria;

A continuación se resumen los comportamiento de esta parte del programa:

- **Si** me encuentro **fuera** de la línea
  - **Si** el **último sensor** en detectar fue el **izquierdo**, **girar** a la **izquierda**
  - **Si no girar** a la **derecha**
- **Si no**, **avanzar**.
-

En la siguiente tabla se remarcan los significados de los valores de las variables “ultimo” y “dentro”.

VARIABLE	VALOR	SIGNIFICADO
dentro	0	El NXT se encuentra **fuera** de la trayectoria.
	1	El NXT se encuentra **dentro** de la trayectoria.
ultimo	0	El último sensor en detectar la línea fue el izquierdo.
	1	El último sensor en detectar la línea fue el derecho.

Para construir la trayectoria, una opción puede ser utilizar cinta negra sobre un piso claro (blanco de preferencia).

Parte 5

Quinto proyecto

“Robot medidor de distancia”

5. Proyecto “Robot medidor de distancia”

Una característica interesante de los servomotores que tiene el NXT, es que pueden proporcionar información sobre su estado, como el ángulo de su posición, cuánto ha rotado, entre otras cosas.

Con esta información se puede medir la distancia recorrida del robot móvil. Sin embargo, y debido a que el NXT no contempla funciones específicas o dedicadas directamente a obtención de medidas en alguna unidad métrica en específico, sino mediante el número de vueltas que ha dado sus servomotores. Será hará necesario entonces primeramente el definir una unidad de medida, la cual se utilizara posteriormente como base para calcular todas las mediciones. Esta unidad de medida debe poder ajustarse a diferentes sistemas métricos y también a diferentes unidades métricas, así entonces por ejemplo será posible obtener la distancia en centímetros, metros o múltiplos de metro, o pies, en el sistema de medidas que se desee.

5.1 Reglas de comportamiento del robot

Para organizar este proyecto se proponen dos etapas de operación, la primera asignara el valor de la unidad de medida. La segunda, comenzara a medir, pero usando siempre como base la unidad de medida definida previamente en la primer etapa.

Para implementar la primer etapa se propone el diseño de una función la cual se activa inmediatamente al iniciar el programa principal. Esta función empezara a activar los motores del NXT en reversa obteniendo un numero de giros en los servomotores de las ruedas del robot. La función se detendrá cuando registre el evento del sensor de tacto (botón). En ese momento asigna como unidad de medida, la distancia recorrida por los motores. Finalmente después desplegara el resultado en la pantalla del NXT.

Para implementar la segunda etapa se utilizara la unidad de medida calculada por la primer etapa e iniciara al tocar el sensor de tacto (registrarse de nuevo el evento sensor de tacto). Una vez iniciada esta etapa, (al tocar el botón del sensor de tacto) los motores giran en reversa y utilizan la unidad de medida calculada en la primera etapa, calculando el numero de veces que se repitió esta unidad. Por ultimo se desplegara el resultado en la pantalla del NXT.

Finalmente cada vez que se presione el boton, el valor medido se borrara y se volvera a repetir la segunta etapa, calculandose un nuevo valor de distancia.

Asi, en pocas palabras, una vez que se haya medido una distancia, el robot se detendrá y marcará la cifra calculada en la pantalla del NXT. Si se quiere medir otra vez, bastará con presionar el botón para que comience el proceso nuevamente.

La función se llamará “medir” y es la siguiente:

FUNCION medir
    MIENTRAS no se presione el sensor **HACER**
        Retrocede
    Detener
    Regresar el valor del motor.
FIN medir

Para la función “main” se tiene:

ACTIVIDAD main
    LLAMAR medir (Calcular unidad de medida)
    REPETIR indefinidamente
        SI se presiona el boton **ENTONCES**
            LLAMAR medir
            CALCULAR el numero de veces que se repitio “unidad de medida” (medicion)
            IMPRIMIR el resultado en pantalla de la distancia medida
    FIN REPETIR
FIN main

5.2 Programa del robot.

Para este programa usaremos centímetros. Es necesario que, para que funcione correctamente el algoritmo, se coloque el robot a 1cm de distancia de algún obstáculo la primera vez. Así el patrón obtenido será de 1cm.

```
#include "NXCDefs.h"
#define PWR 50

void retrocede(){
  OnRev( OUT_B, PWR );
  OnRev( OUT_C, PWR );
}
```

```
void detener(){
  Off( OUT_BC );
}

long medir(){
  long m;
  SetSensorTouch( IN_1 );

  while( Sensor( IN_1 ) == 0 ){ //Mientras no se presione el boton
    retrocede();
    TextOut( 20, 20, "Midiendo...", true );
  }
  detener();

  return MotorRotationCount( OUT_B );
}

task main(){
  long patron;
  long medida;

  SetSensorTouch( IN_1 );
  patron = medir();
  TextOut( 20, 20, "Establecido", true );
  while( Sensor( IN_1 ) );

  while( true ){
    if( Sensor( IN_1 ) == 1 ){
      while( Sensor( IN_1 ) );

      ResetRotationCount( OUT_B );
      medida = medir();

      NumOut( 0, 0, medida/patron, true );
      TextOut( 25, 0, "cm", false );

      while( Sensor( IN_1 ) );
    }
  }
}
```

5.3 Explicación del código

t.

```
long medir(){
  long m;
  SetSensorTouch( IN_1 );
```

```
  while( Sensor( IN_1 ) == 0 ){ //Mientras no se presione el
boton
      retrocede();
      TextOut( 20, 20, "Midiendo...", true );
  }
  detener();

  return MotorRotationCount( OUT_B );
}
```

Función “medir”. Mientras no se presione el boton, el robot seguirá retrocediendo. Al final regresa la distancia en las unidades del robot. Las funciones nuevas en este código son “TextOut” y “MotorRotationCount”.

TextOut imprime texto en la pantalla. En este caso es usado simplemente para mostrar el estado en el que se encuentra el programa.

MotorRotationCount es la función que nos devuelve lo que ha recorrido el motor, pero en otras unidades.

Dentro del main hay que destacar lo siguiente:

```
patron = medir();
TextOut( 20, 20, "Establecido", true );
while( Sensor( IN_1 ) );
```

Aquí se mide el patrón y se asigna a una variable. La razón del while es para evitar que el proceso continúe hasta que se ha retirado al robot de la pared o del objeto. Es decir, espera hasta que el boton haya sido levantado.

```
while( true ){
    if( Sensor( IN_1 ) == 1 ){
      while( Sensor( IN_1 ) );

      ResetRotationCount( OUT_B );
      medida = medir();

      NumOut( 0, 0, medida/patron, true );
      TextOut( 25, 0, "cm", false );

      while( Sensor( IN_1 ) );
    }
  }
```

Este bloque estará realizando mediciones constantemente.
Se apoya en la función “medir”.

ResetRotationCount inicializa el contador de la distancia del NXT. Es importante reiniciarlo para cada medición.

Los while cerrados, son candados de espera para evitar que se hagan mediciones no esperadas.

Parte 6

Sexto proyecto

“Robot comandado por sonido”

6. Proyecto robot comandado por sonido.

Se puede tener una vía de comunicación alternativa con el NXT, la cuál no esta relacionada con tecnologías como el bluetooth ni el cable USB: El sonido.

Se pueden crear varias aplicaciones en las cuáles se le envíe información al robot mediante sonido. En esta práctica lo usaremos para dar instrucciones sobre la dirección y el comportamiento del robot móvil por medio de aplausos.

6.1 Reglas de comportamiento del robot.

Para identificar un aplauso, lo primero que se podría pensar es que el sensor de sonido nos marque un valor muy alto y luego uno muy bajo, lo cual es verdad. Entonces:

Si el sonido es alto
    Esperar a que el sonido baje
Si el sonido baja después de que pasó por alto
    Hubo un aplauso

Si queremos contar más de un aplauso, podemos simplemente declarar un contador e irlo aumentando cada vez que se genere un aplauso. Mientras más aumente la cantidad de aplausos generados, mas retardada será la respuesta del robot dado que necesitará esperar cierto tiempo para saber, por ejemplo, si era solo un aplauso o si habrá más.

Para evitar que la respuesta sea tan tardía, usaremos simplemente dos comandos: Un aplauso y dos aplausos. Su interpretación será la siguiente:

Señal	Interpretacion
1 Aplauso	Avanzar ó Girar a la izquierda
2 Aplausos	Detenerse

Para fines didácticos, se separarán en dos actividades el programa. En una se contarán los aplausos (sonidos), mientras que en la otra se tomará una decisión con respecto a éstos (main).

6.2 Programa del robot

El programa es el siguiente:

```
#include "NXCDefs.h"
#define ALTO 25
#define AVANZAR 1
#define PWR 50
#define ESPERA 200

int senal;

void avanzar(){
	OnFwd( OUT_BC, PWR );
}
void girar_izq(){
	OnFwd( OUT_B, PWR-25 );
	OnRev( OUT_C, PWR-25 );
}
void detener(){
	Off( OUT_BC );
}

task sonido(){
	int alto = 0;
	SetSensorSound( IN_2 );

	while( true ){
		if( Sensor( IN_2 ) > ALTO ){
			alto = 1;
		} else if( alto == 1 ){
			senal++;
			alto = 0;
		}
		NumOut( 0, 0, senal, false );
	}
}

task main(){
	start sonido;

	int instrucc = 0;
	senal = 0;
```

```
        while( true ){
                switch( senal ){
                        case 0:
                        break;
                        case AVANZAR:
                                Wait( ESPERA );
                                if( senal > 1 )
                                        break;
                                if( instrucc == 0 ){
                                        avanzar();
                                        instrucc = 1;
                                } else {
                                        girar_izq();
                                        instrucc = 0;
                                }
                                senal = 0;
                        break;
                        default:
                                detener();
                                senal = 0;
                        break;
                }
        }
}
```

6.3 Explicación del código.

```
#include "NXCDefs.h"
#define ALTO 25                 //Indica el umbral donde un sonido se
considerará fuerte o no.
#define AVANZAR 1                       //Cantidad de aplausos que
indicarán avanzar o girar.
#define PWR 50                  //Potencia de los motores.
#define ESPERA 200              //Tiempo de espera para el siguiente
aplauso.

int senal;                      //Contador de aplausos

void avanzar(){
        OnFwd( OUT_BC, PWR );
}
void girar_izq(){
        OnFwd( OUT_B, PWR-25 );
        OnRev( OUT_C, PWR-25 );
```

```
}
void detener(){
	Off( OUT_BC );
}
```

Toda esta sección ya debe ser familiar para el lector. Se declaran las constantes:

ALTO – Indica el umbral donde un sonido se considerará fuerte o no.

AVANZAR – Cantidad de aplausos que indicarán avanzar o girar.

PWR – Potencia de los motores.

ESPERA – Tiempo de espera para el siguiente aplauso.

Luego se declara la variable global **senal**, la cuál nos servirá como contador de aplausos. Después vienen las 3 instrucciones básicas que usaremos en el proyecto.

```
task sonido(){
	int alto = 0;
	SetSensorSound( IN_2 );

	while( true ){
		if( Sensor( IN_2 ) > ALTO ){
			alto = 1;
		} else if( alto == 1 ){
			senal++;
			alto = 0;
		}
		NumOut( 0, 0, senal, false );
	}
}
```

Esta actividad se dedicará exclusivamente a contar los aplausos que escuche el robot.

La función que tiene la variable alto, es la de clasificar el sonido que entra al sensor. Cuando vale 1, el sonido es alto y 0 de lo contrario.

Una vez que alto vale 1, entonces esperamos a que el sonido baje. Cuando baja, se aumenta senal, para contar otro aplauso. Entonces alto se regresa a 0 para esperar.

Lo nuevo aquí es la siguiente línea.

```
		NumOut( 0, 0, senal, false );
```

Esta imprime en pantalla, en la posición (0, 0), el valor de senal.

```
task main(){
	start sonido;

	int instrucc = 0;			//Variable para alternar
entre avanzar y girar_izq
	senal = 0;

	while( true ){
		switch( senal ){
			case 0:
			break;
			case AVANZAR:
				Wait( ESPERA );		//
Espera por si llega un segundo aplauso.
				if( senal > 1 )
					break;
				if( instrucc == 0 ){		//Aqui
se decide si avanzar o girar.
					avanzar();
					instrucc = 1;
				} else {
					girar_izq();
					instrucc = 0;
				}
				senal = 0;
			break;
			default:
				detener();
				senal = 0;
			break;
		}
	}
}
```

La actividad “main” se centra principalmente en la toma de decisiones, como ya se mencionó anteriormente. Comienza ejecutando la actividad “sonido”.

La variable “instrucc”, servirá de bandera para ir alternando entre “avanzar” y “girar_izq”. De manera que un aplauso provoque que si el robot estaba avanzando, entonces gire, y viceversa.

El switch simplemente es la toma de decisiones:

Si no hubo aplausos...

```
case 0:
break;
```

Si hubo un aplauso...

```
case AVANZAR:
        Wait( ESPERA );                 //
Espera por si llega un segundo aplauso.
        if( senal > 1 )
                break;
        if( instrucc == 0 ){            //Aqui
se decide si avanzar o girar.
                avanzar();
                instrucc = 1;
        } else {
                girar_izq();
                instrucc = 0;
        }
        senal = 0;
break;
```

En otro caso...

```
default:
        detener();
        senal = 0;
break;
```

Cabe destacar:

```
        Wait( ESPERA );                 //
Espera por si llega un segundo aplauso.
        if( senal > 1 )
                break;
```

La función Wait esperará a un posible segundo aplauso, porque si no esperara, jamás lo contaría y siempre habría solamente un solo aplauso.

Ahora, si hubo un segundo aplauso, se debe cancelar todo el demás procedimiento para que pase al segundo caso. Ese es el motivo del "if".

También es importante observar que cada vez que main ejecuta una instrucción, inicializa senal a '0' (cero), para comenzar el conteo nuevamente.

Parte 7

Septimo proyecto

“Robot seguidor de objetos”

7. Proyecto robot seguidor de objetos.

Teniendo solamente un sensor de ultrasonido es algo complicado pensar en cómo seguirle la pista a algún objeto que se detecte mediante este sensor. Dado que una vez que se pierde el objeto de la vista del robot es imposible determinar la dirección hacia donde se movió (derecha o izquierda), solamente será posible seguirlo a prueba y error.

7.1 Reglas de comportamiento del robot.

El comportamiento que deberá tener el robot una vez que se pierde al objeto de vista, será buscarlo en una dirección. De no encontrarlo, comenzará a buscarlo en la otra. Si ya no lo encuentra en la otra tampoco, continuará buscando en esa misma (la última) ya que no tiene caso que regrese a la dirección anterior. Así continuará girando hasta que detecte algún objeto.

Entonces:

SI detecta algún objeto **ENTONCES**
    AVANZAR
SINO SI continúa sin detectar al objeto **ENTONCES**
    BUSCAR por la izquierda
SINO SI continúa sin detectar al objeto **ENTONCES**
    BUSCAR por la derecha

SI ya alcanzó al objeto **ENTONCES**
    DETENER

7.2 Programa del robot

```
#include "NXCDefs.h"
#define PWR 75
#define ALCANZADO 30

void avanzar(){
    OnFwd( OUT_BC, PWR );
}
void detener(){
    Off( OUT_BC );
}
void girar_izq(){
    OnFwd( OUT_C, PWR-30 );
    OnRev( OUT_B, PWR-30 );
```

```
}
void girar_der(){
    OnFwd( OUT_B, PWR-30 );
    OnRev( OUT_C, PWR-30 );
}

task main(){
  SetSensorLowspeed( IN_4 );
  int us;
  while( true ){
      if( SensorUS( IN_4 ) > ALCANZADO && SensorUS( IN_4 )
!= 255 ){
            detener();
            Wait( 200 );
            avanzar();
            while( SensorUS( IN_4 ) > ALCANZADO &&
SensorUS( IN_4 ) != 255 );
            ResetRotationCount( OUT_C );
      }
      else if( SensorUS( IN_4 ) == 255 && MotorRotationCount(
OUT_C ) < 200 ){
            girar_izq();
            ResetRotationCount( OUT_C );
            while( SensorUS( IN_4 ) == 255 &&
MotorRotationCount( OUT_C ) < 200 );
      }
      else if( SensorUS( IN_4 ) == 255 ){
            girar_der();
            while( SensorUS( IN_4 ) == 255 );
            ResetRotationCount( OUT_C );
      }
      else if( SensorUS( IN_4 ) <= ALCANZADO ) {
            detener();
            while( SensorUS( IN_4 ) <= ALCANZADO );
            ResetRotationCount( OUT_C );
      }
  }
}
```

7.3 Explicación del código

```
      if( SensorUS( IN_4 ) > ALCANZADO && SensorUS( IN_4 )
!= 255 ){
            detener();
            Wait( 200 );
            avanzar();
```

```
            while( SensorUS( IN_4 ) > ALCANZADO &&
SensorUS( IN_4 ) != 255 );
            ResetRotationCount( OUT_C );
      }
```

Esta sección es la que se refiere a cuando el robot detecta algún objeto y debe acercarse a él. Obsérvese que el valor del sensor se compara con 255. El 255 indica que no se está recibiendo señal de ningún objeto. En este caso debe ser distinto a 255, ya que se supone se está detectando algún objetivo.

La razón de que se llame a la función detener( ) es para dar mayor precisión.

El while anidado no realiza ninguna acción en sí, simplemente detiene la ejecución del programa hasta que la condición sea falsa.

La función ResetRotationCount(OUT_C) regresa el valor de giros del motor C a cero.

```
      else if( SensorUS( IN_4 ) == 255 && MotorRotationCount(
OUT_C ) < 200 ){
            girar_izq();
            ResetRotationCount( OUT_C );
            while( SensorUS( IN_4 ) == 255 &&
MotorRotationCount( OUT_C ) < 200 );
      }
```

El programa entra en este bloque cuando dejó de detectar al objeto. En esta parte, comenzará a buscar por la izquierda. Aquí se da importancia al giro del motor C dado que en base a este, se establece qué tanto buscará por la izquierda.

```
else if( SensorUS( IN_4 ) == 255 ){
            girar_der();
            while( SensorUS( IN_4 ) == 255 );
            ResetRotationCount( OUT_C );
      }
```

Esta parte se ejecuta si no encontró nada por la izquierda.

```
 else if( SensorUS( IN_4 ) <= ALCANZADO ) {
            detener();
            while( SensorUS( IN_4 ) <= ALCANZADO );
            ResetRotationCount( OUT_C );
      }
```

Aquí el robot ha alcanzado a su objetivo y se detiene. Espera a que el objeto se mueva.

Apendice

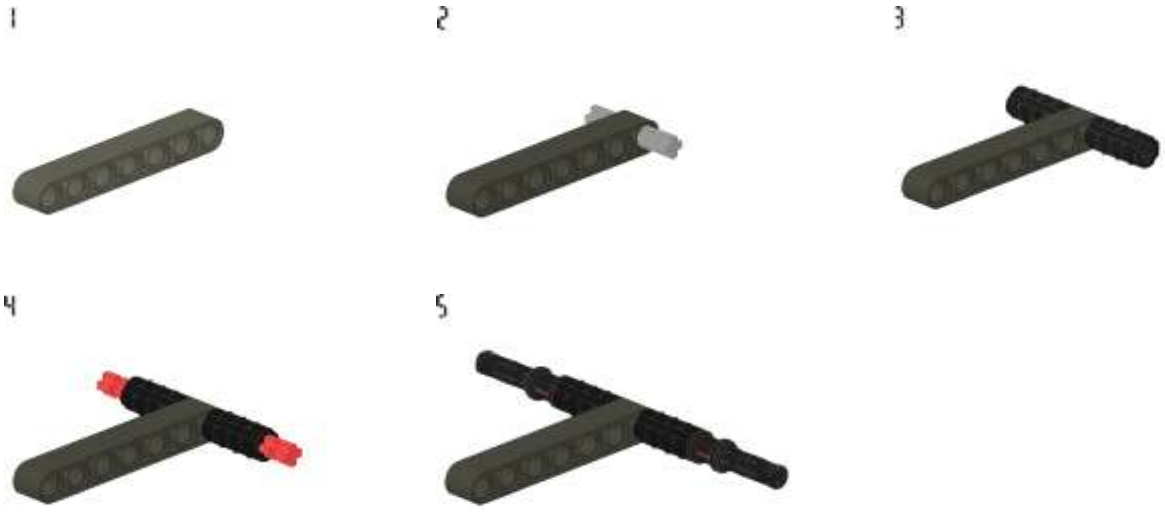
A

Construccion del robot “Explorador”

A continuación se presentaran imágenes secuenciales que indicaran gráficamente el orden construccion de cada parte del robot rodante presentado en la parte 1 de este material.

Al final se integran todas estas partes para conformar el robot. La secuencia u orden de construccion se indica mediante un numero en la parte superior izquierda

Primero se explicara la construccion de la parte 1:



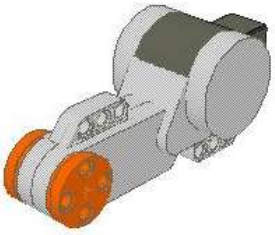
Ahora se detalla la construccion de la parte 2:

1

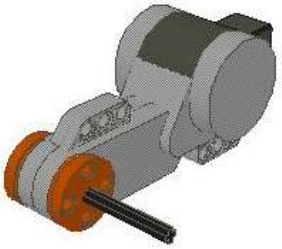
2

Ahora se detalla la construccion de la parte 3:

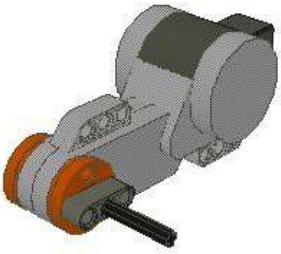
1



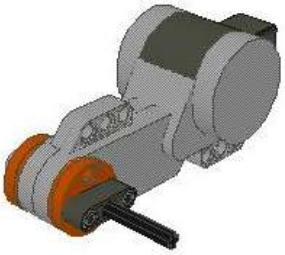
2



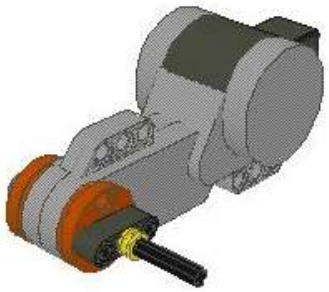
3



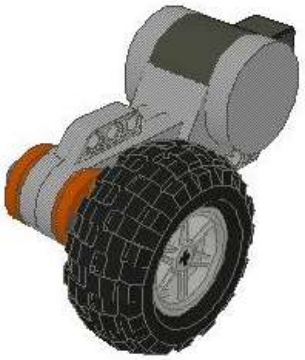
4



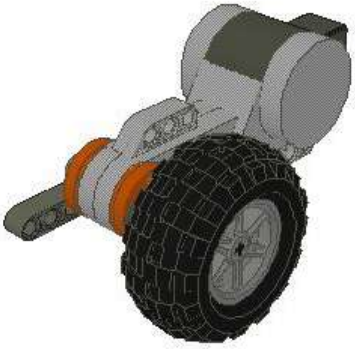
5



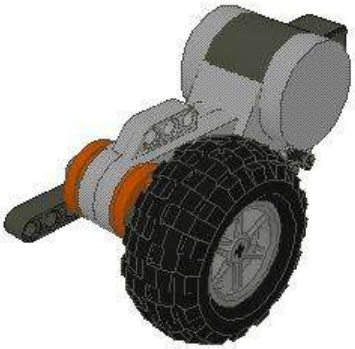
6



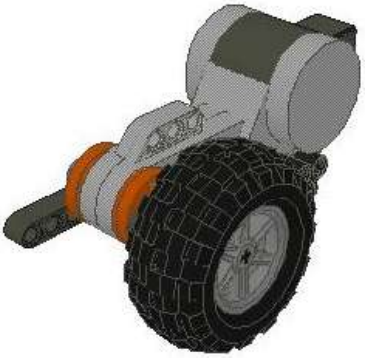
7



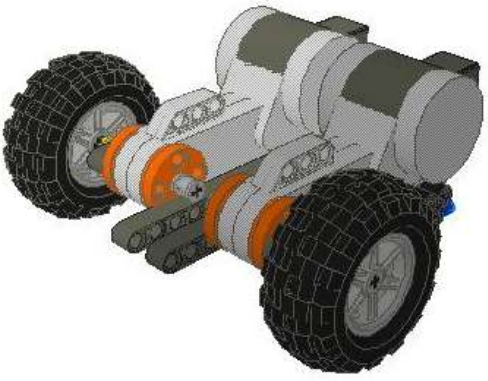
8



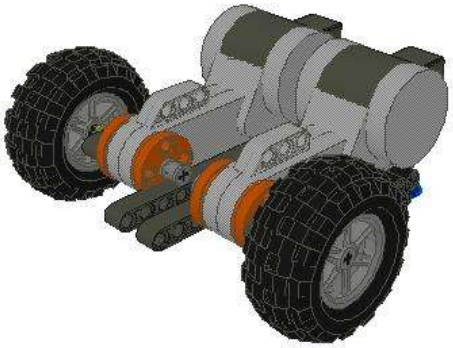
9



10



11

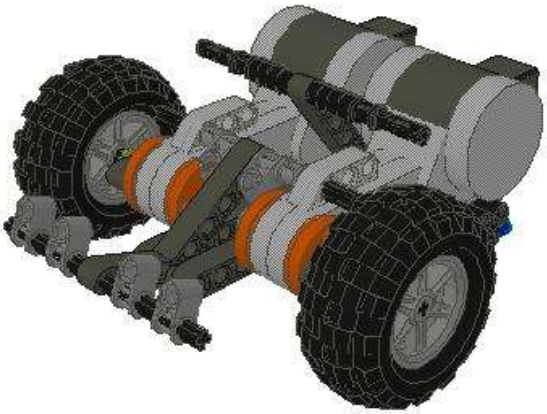


En el paso siguiente se integra la parte 2 (previamente construida) con la estructura de dos ruedas

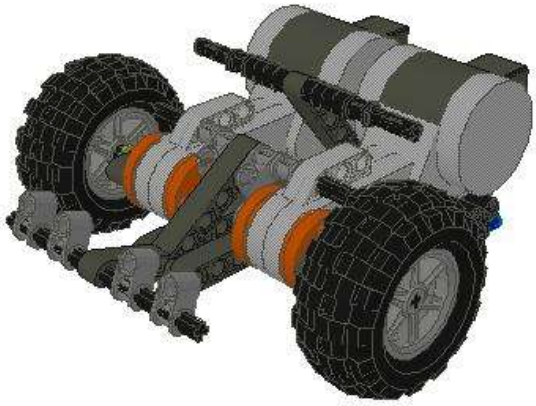
14

15

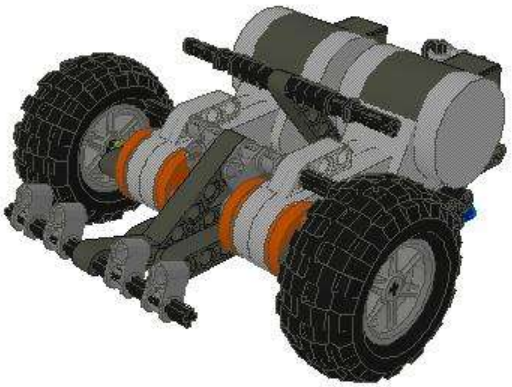
16



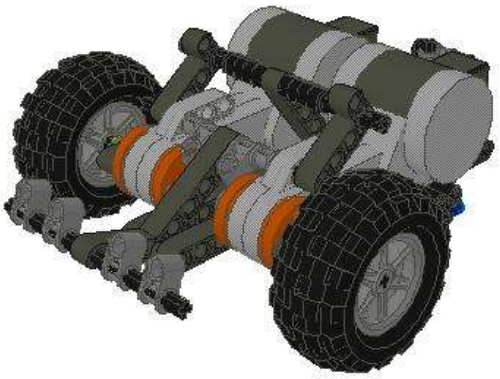
17



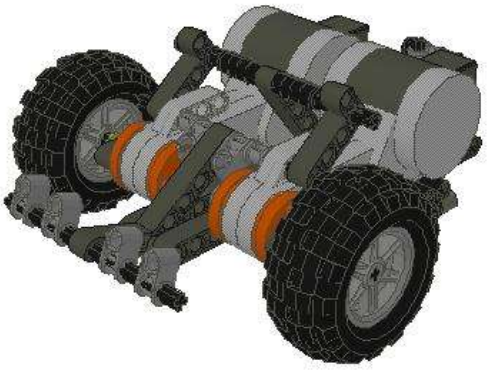
20



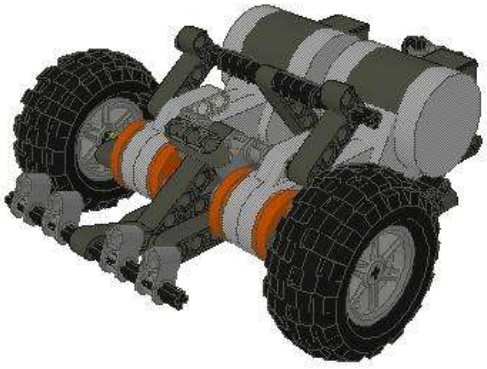
22



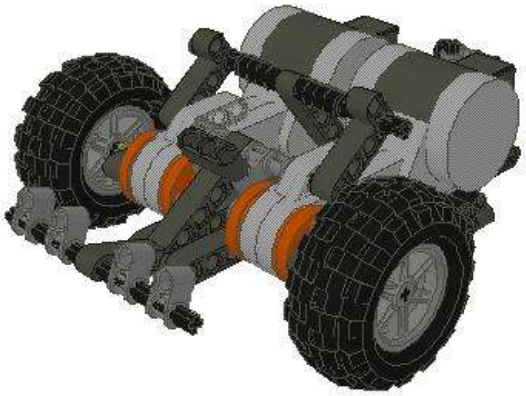
23



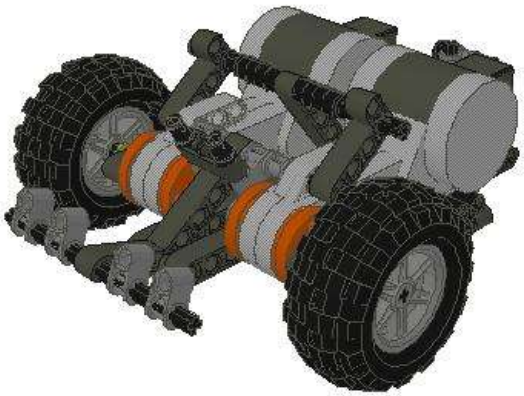
24



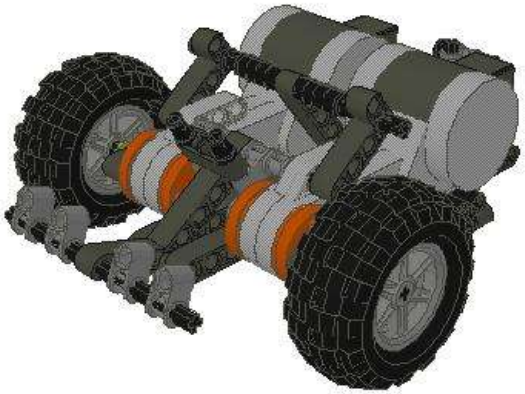
25



26

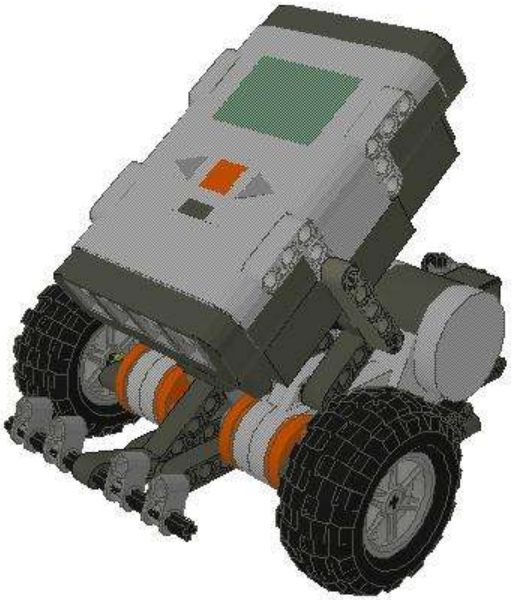


27

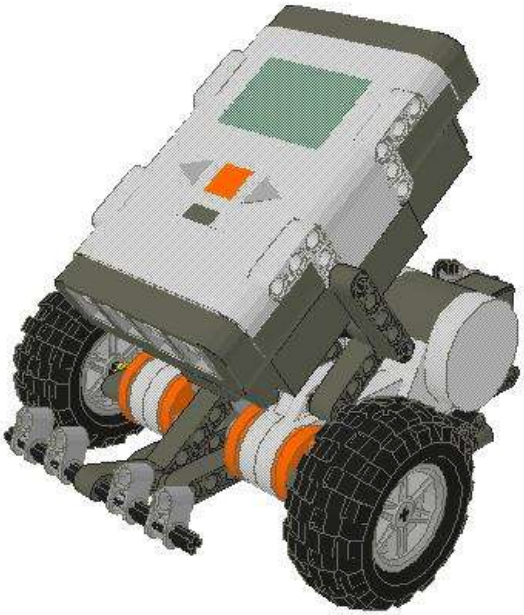


28

29



30



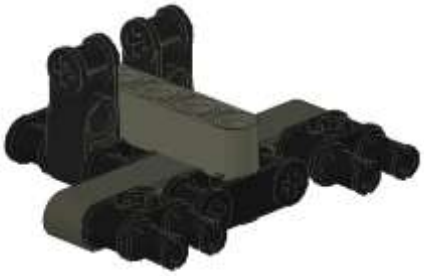
Ahora continuaremos con la construccion de la parte 4:

1

4

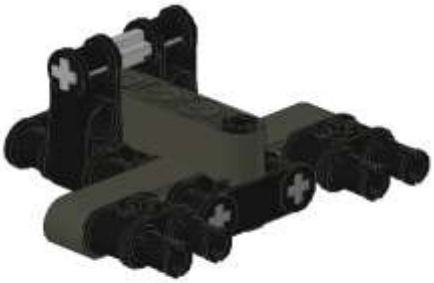
5

7

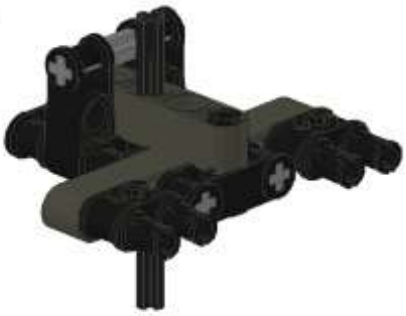


9

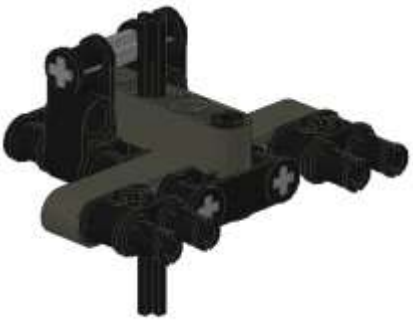
10



11



12



13

14

15

16

* Girando la pieza obtenida en el paso 19 tenemos

Apendice

B

Construccion del “Robot sigue lineas”

El carro de robot a utilizar es el mismo descrito en el apendice A, solo se agregaran la parte del sensor de luz infrarroja.

1

2

3

4

6

7

8